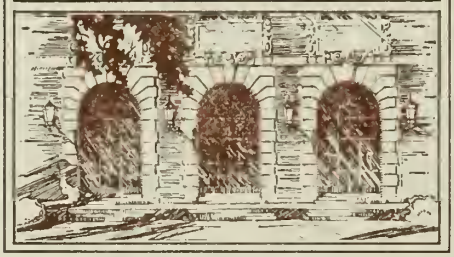


UIUCDCS-R-76-812

A MICROPROCESSOR-CONTROLLED INTERFACE FOR BURST PROCESSING

by

ROBERT MILTON PLEVA

July, 1976

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

UIUCDCS-R-76-812

A MICROPROCESSOR-CONTROLLED INTERFACE FOR BURST PROCESSING

BY

ROBERT MILTON PLEVA

July 1976

Department of Computer Science
University of Illinois
Urbana, Illinois 61801

This work was supported in part by Contract No. USNAVY N0014-75-C-0982 and was submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science, at the University of Illinois.

ACKNOWLEDGEMENT

The author wishes to express his gratitude to his thesis advisor, Professor Michael Faiman, and to IEL Director W. J. Poppelbaum for their support and guidance in this effort. Thanks are also due to Mrs. Kathy Gee for her help with the manuscript, and to the staff of the Information Engineering Laboratory for their valued friendship.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

1.0 Background

Burst Processing represents a new concept in the encoding, transmission, and processing of analog-derived data.[1] Numerous hardware prototypes have demonstrated the applicability of this technique to a variety of problem areas--e.g., arithmetic processing, audio and video-bandwidth data transmission, audio noise suppression, and even AM and FM demodulation.[2,3,4]

The Burst technique is basically a combination of two elementary notions, namely, that of a unary (radix one) encoding of digital data, and the principle of averaging to obtain increased precision (as in stochastic processing). A more detailed description of the Burst representation will be presented in the next section, but suffice it to say that the aforementioned characteristics suggest the use of Burst processing for applications such as shipboard communications links where wide-bandwidth transmission channels (e.g., coaxial cables or optical fibers) are available and some degree of noise tolerance is a major requirement.

As desirable as Burst processing is for a certain class of problems, it is clear that the weighted-binary codings traditional in digital computers are still of primary importance for generalized numerical processing and essentially all processing of non-numeric (e.g., control) information. Demonstration of some degree of compatibility, therefore, between the Burst and weighted-binary modes of data representation is essential if Burst processing is to achieve the status of a viable and useful technique. This is not to argue that Burst systems must always depend upon the proximity of a classical digital system to be useful--indeed a great deal of autonomous control can be provided for in the Burst hardware itself, including magnitude comparison and ranging, peak detection, and error correction. But certain

elementary functions--data storage being a prime example--are not well-suited to the Burst representation due to the low information content/bit inherent in the unary encoding. For these functions, traditional techniques continue to be required. Thus, for a very complex system as is represented by the total electronics of a modern naval vessel, compatibility between all modes of data representation, whether analog, binary, stochastic, or Burst, is essential. Such compatibility is necessary to provide all desired functions, as already argued, and provides additional system integrity by allowing cross-monitoring between various subsystems to check for proper operation, and by providing redundant systems to reduce the impact of isolated hardware failures.

The MICROBURST project was undertaken to investigate this question of compatibility, and to construct hardware which would allow the interconnection of a hypothetical classical computer system to a multichannel Burst network. The goal was not, of course, to define a completely general-purpose interface, since experience has shown that such "universal" solutions rarely, if ever, fulfill their claim. Instead, the goal has been to define a structure, along with the elementary hardware required for most applications, which would greatly ease the problem of interfacing particular systems. Greatest flexibility and hardware simplicity has been obtained through the use of a low-cost microprocessor as control element for the interface system.

1.1 The BURST Representation

The most fundamental notion of Burst processing is that of transmitting a sequence of digits in some radix in such a manner that the average of these digits corresponds to the value of the datum being represented. Of course, since finiteness of the data stream is a practical necessity, the precision of the representation is limited by both the encoding and decoding processes. The

attractiveness of the resulting digit-serial representation lies in the possibility of doing on-line manipulation of such data using very simple (i. e., essentially one digit) processing elements.

A secondary, albeit very important, concept of Burst processing is the unary encoding used to represent the "digits" in the data stream. Such a unary code obviously requires a number of bits (or time slots, in a serial system) equal to the radix; the resulting bit inefficiency is a disadvantage, but the greatly decreased sensitivity to single-bit errors must also be considered when comparing with standard weighted-binary codes.

For convenience, the "radix" assumed when discussing Burst processing systems has come to be standardized at ten. Thus the Burst digit (or "frame" as it is often called) consists of ten time slots during each of which either a pulse (corresponding to a unit) or no pulse is transmitted. If all of the units to be transmitted in a given frame occur in a string immediately after the beginning of the frame, the frame is said to be in "packed Burst format." If, on the other hand, units are scattered at random throughout the frame, the resulting format is termed "unpacked." A schematic representation of these two possible formats is shown in Figure 1. Although the mere representation of data favors neither format, most important processing functions require the packed format, and this has become the standard technique.

In Figure 1, the Burst data is shown in a Return-to-Zero (RZ) format; that is, each data bit is separated from adjacent bits by time intervals during which the data line is spacing (i.e., at logical "0"). This format makes oscilloscope interpretation of data streams more convenient, but NRZ (Not-Return-to-Zero) coding may be used equally well. Either of these techniques requires that clocking information be provided on a separate line. Any of

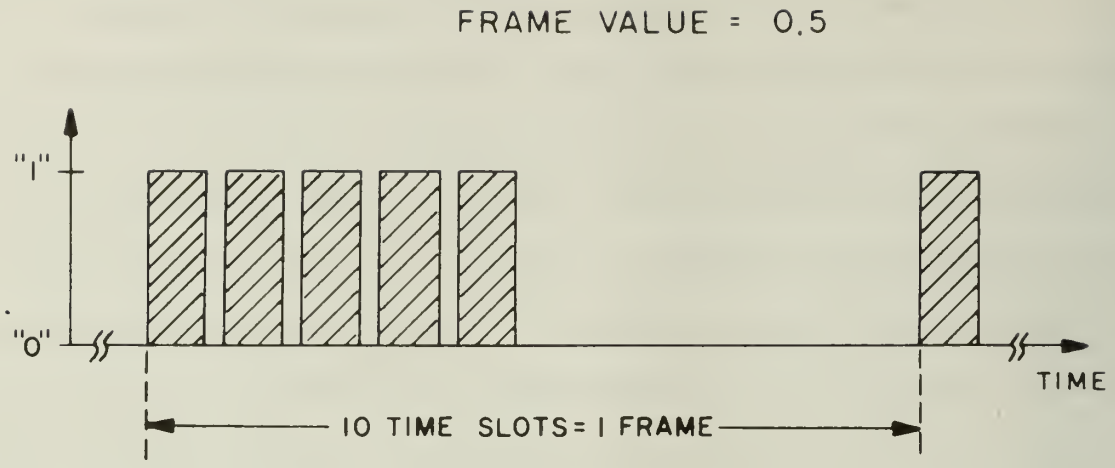


Figure 1a Packed Burst Format

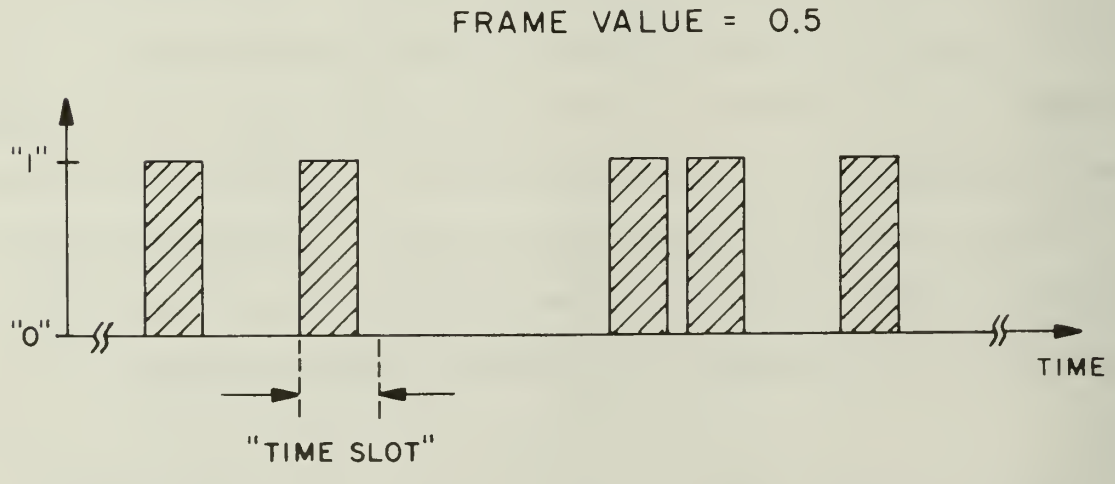


Figure 1b Unpacked Burst Format

the standard self-clocked codings such as phase or frequency modulation may be used to eliminate the requirement for a separate clock line, although these techniques have found little use to date.

The Burst representation of negative numbers is another problem which may be approached in several ways. The most straightforward technique is a "sign-plus-Burst-magnitude" scheme which again requires an additional line (carrying sign information) for each Burst data line. This approach has actually been used in at least one Burst prototype[2], but the need for parallel signal lines in an inherently serial system is highly undesirable. Another possibility is bipolar modulation, wherein positive-going pulses (say) indicate magnitudes greater than zero, and negative-going pulses define negative values. The most direct approach, however, is the biased representation which maps a signal magnitude of zero onto half-filled Burst frames. This technique has the advantages of requiring no special hardware to couple the logic circuits to the channel, and of requiring no additional signal lines as well. Thus, this mapping has become the favored approach when representation of negative values is necessary.

1.2 BURST Generation Techniques

In order to better understand some of the unique advantages of the microprocessor-controlled interface, it will be useful to consider the standard techniques used for generating Burst data from analog sources. Note that for a Burst stream to accurately represent an analog variable, it is necessary that the analog information be effectively at DC with respect to the averaging time needed to obtain the desired precision. Now the averaging time needed to obtain r decimal digits of precision from any deterministic unary-encoded data stream operating at a bit rate of f_d Hertz is clearly:

$$t_r = \frac{10^r}{f_d} \qquad \text{[Eq. 1.2]}$$

Since the maximum value of the derivative of $\sin(2\pi ft)$ is $2\pi f$, we can insure that the waveform represented will change by a relative amount less than 10^{-r} if the highest frequency component of the input signal, f_M, satisfies:

$$2\pi f_M t_r \leq 10^{-r}$$

Thus,

$$f_M \leq \frac{f_d}{2\pi \cdot 10^{2r}}$$

Note that this is an exceedingly severe restriction, since it implies that for a precision of three decimal digits (.1 percent) and a data rate of f_d=1 MHz, the highest frequency component of the input signal must be limited to $f_M \leq \frac{1}{2\pi}$ Hz $\cong$ .15 Hz. This limitation applies only when the analog variable is presented directly to a Burst encoder without the benefit of sample-and-hold circuitry. If S/H hardware is included, of course, then the bandwidth restriction is defined only by the Nyquist criterion requiring two samples per cycle of the highest frequency component, and the time necessary to output this data to the required precision (t_r). In this case, it is easy to see that:

$$f_{M,S/H} \leq \frac{f_d}{2 \cdot 10^r}$$

Therefore, if an S/H-equipped encoder is used with the figures of the previous calculations, we find that f_M need only be limited to 500 Hz or less--a bandwidth increase of greater than three orders of magnitude. The conclusion, then, is that sample-and-hold circuitry is a practical necessity for high precision representation of even audio-frequency signals due to the very long time needed to encode the value. We shall assume, then, that these

bandwidth restrictions are adhered to as we describe the encoding techniques in the following paragraphs.

Figure 2 demonstrates the operation of the simplest Burst encoder based upon a staircase waveform generated by shifting ones into a Block Sum Register. [Note: A Block Sum Register, or BSR, is simply a shift register with each bit position connected to a current source; the currents are summed on a bus and converted to a voltage output. It is thus simply an unweighted (unary) digital-to-analog converter with a shift register.] The encoder merely outputs a one during each bit time until the staircase voltage exceeds the analog input signal. When completely filled, the register is direct cleared and the cycle repeats. The encoder thus produces a continuous Burst stream, which is however accurate to only 10 percent (the resolution of a single frame) since successive frames represent different samples of the input signal. In this case averaging over a number of frames does not necessarily increase the precision of the result.

A straightforward extension of this scheme called the "vernier encoder"[1] can be used when greater precision is required. Essentially a cascade of simple encoders having a decade relationship between fullscale output currents, the circuit operates as illustrated in Figure 3. Although the circuit is simple and produces a correct stream average, its output does not converge to the correct result as quickly as possible as can be seen by considering the encoder output for a signal value of 3.2. The output stream in this example consists of 2 frames of value 4 followed by 8 frames of value 3, at which point the "ramp" registers are cleared and the cycle repeats. Fastest convergence, however, would clearly be provided if the 10-frame output sequence were 3343333433. While such an optimal decomposition is very difficult to

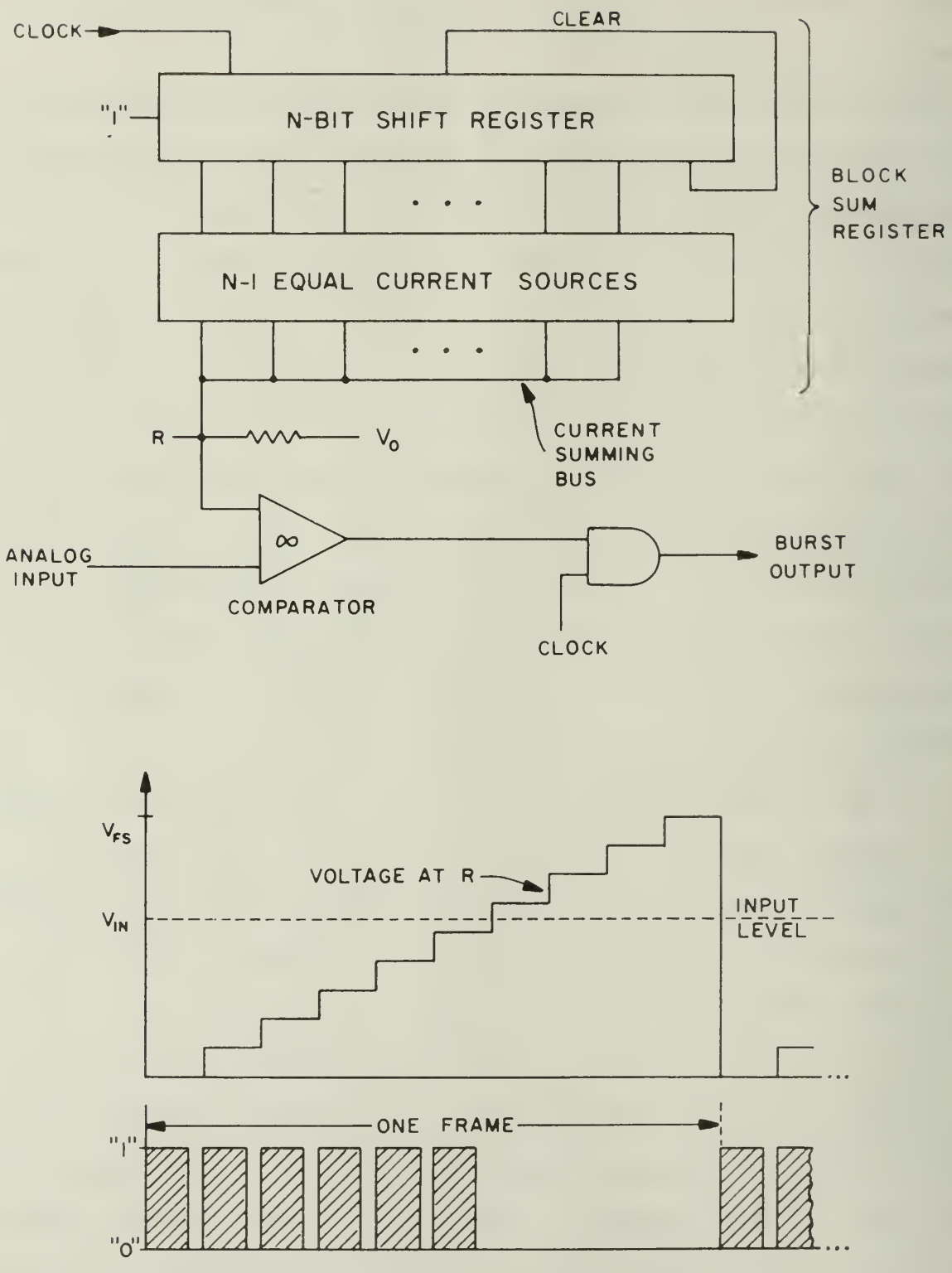
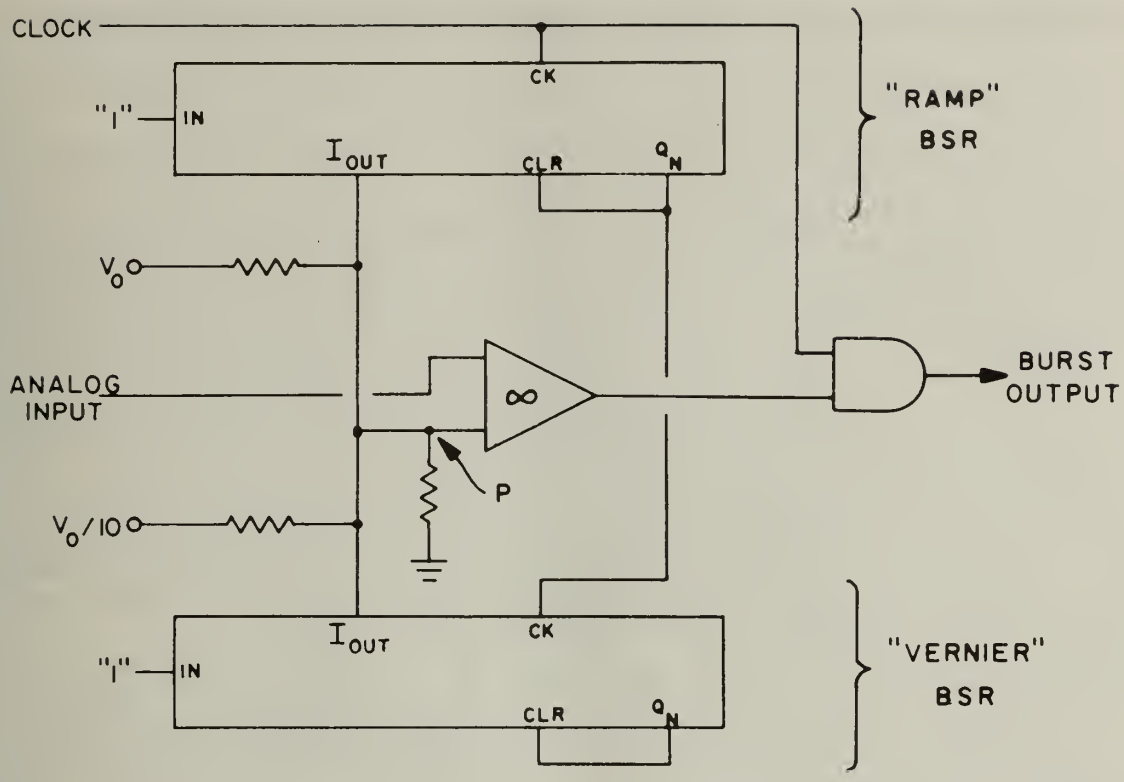


Figure 2 Stairstep Burst Encoder



EXAMPLE OF VOLTAGES AT P:

1ST BLOCK	2ND BLOCK	3RD BLOCK	. . . . .	10TH BLOCK
.00	.01	.02	. . . . .	.09
.10	.11	.12	. . . . .	.19
.20	.21	.22	. . . . .	.29
.30	.31	.32	. . . . .	.39
.40	.41	.42	. . . . .	.49
.50	.51	.52	. . . . .	.59
.60	.61	.62	. . . . .	.69
.70	.71	.72	. . . . .	.79
.80	.81	.82	. . . . .	.89
.90	.91	.92	. . . . .	.99
4	4	3	. . . . .	3

Figure 3 Two-Decade Vernier Encoder

obtain from simple analog-to-Burst generation circuitry, the MICROBURST interface provides Burst outputs having this property, as will be discussed in more detail in later sections.

1.3 The Need for "Intelligent" Control

As was briefly mentioned earlier, the MICROBURST interface incorporates a small computer system--built around a standard microcomputer chip--functioning as a control element. Since the inclusion of such an "intelligent" control device adds considerable cost and complexity to the hardware, it is necessary to define just how its presence will enhance the performance of the final system. Some of the more obvious and important justifications for the microprocessor-based architecture are discussed in this section.

Since Burst-encoded data is represented by bit streams which are, in principle at least, doubly infinite in time scale, it is usually necessary that each variable be transmitted along a separate wire. Various time-division multiplexing schemes can be used to conserve data lines, but such methods tend to negate the most important feature of Burst processing, which is circuit simplicity. Also, such multiplexing demands that synchronization be rigorously maintained which again contradicts the inherently synchronization-free nature of the Burst representation. Ideally, one would like to use a smaller number of wires, each carrying only a Burst value which is needed for processing at that time. In this way, the demultiplexing problem is reduced to merely activating the appropriate processing devices after the corresponding data has been established on the lines. There is thus an immediate need for <u>control</u> in such a system to perform the channel switching functions and to enable the appropriate processing subsystems.

A related problem is that of reconfiguring a Burst system to perform tasks other than those for which it was originally designed. Here again the most flexible solution is to partition the system into an array of Burst Processing Units (hereafter referred to as BPU's) connected by data lines carrying Burst-encoded information. If these data lines can be switched as mentioned above and the operation of the various BPU's controlled by a programmed device, then the entire system can be readily configured to perform any functions for which the appropriate BPU's are available.

Then, too, it must be recognized that there are many operations which are more readily performed by traditional computer methods than by constructing equivalent BPU's. Examples of such operations are: nonlinear transformations, statistical analyses, evaluation of complex functions, and long-term storage. Certain processing systems may tend to require large numbers of simple arithmetic operations and only very few special operations of the type just described. Systems of this type may benefit from the simplicity of performing arithmetic on Burst-encoded data while requiring only limited processing (for the special functions) from a traditional computer.

Furthermore, as was briefly alluded to in the previous section, it is possible to use the processing capability of a small computer to advantage in actually improving the characteristics of generated Burst streams. In particular, we can "arrange" the frame values in such a way as to maximize the rate of convergence of the stream average towards the exact value to be represented. This may or may not be a significant advantage in any particular application (it will be of value only if the stream average is formed sequentially rather than all at once, as can be done by using "long" BSR's or "perverted-adder"[5] summing trees), but it is easily obtained if a small computer is available.

A final point, but one which is especially important until the details of the Burst representation have become firmly standardized, is the flexibility in the interpretation of Burst-encoded data afforded by the inclusion of a classical processing element. In particular, the various approaches to the representation of negative numbers discussed earlier can all be accommodated with ease if the received data is interpreted by a programmed device. The use of differing bias (offset) values then requires only a slight program modification; even different representations within a single system could be handled if so desired.

The high-level of control over hardware functions afforded by the use of a classical processor allows the interface device to act, in effect, as a general "Burst simulator". What this means is that even networks which are not required to interface to computer systems in normal operation can be debugged and evaluated "off-line" by using the interface to represent the "Burst world" within which the network is designed to operate. In this sense the device is useful as "programmed instrumentation" in the development of Burst hardware.

This is only a partial list of the benefits to be derived from including a small computer in the hardware of a general-purpose Burst interface system. These advantages can be largely summed up in a single word--versatility. Any procedure which can be programmed can be readily implemented in such a system. This includes, but is obviously not limited to, such desirable capabilities as automatic scanning of channels, checking against predefined limits, dynamic reconfiguration, etc. Thus the inclusion of an "intelligent" control mechanism--in this case a one-chip microprocessor--is a powerful addition to the interface hardware.

2 SYSTEM STRUCTURE AND CAPABILITIES

2.0 General

The MICROBURST interface was designed to provide as much capability as possible with existing hardware while allowing other functions to be implemented with a minimum of additional effort. The basic architecture adopted follows the general form shown in Figure 4 (a); a number of BPU's (in this case 5 are initially provided) interfacing external Burst data channels, with each BPU's activities controlled by a Central Processing Unit. To realize the goal of simplified substitution of other BPU's, a common control bus scheme was utilized. The control bus layout and protocol actually used borrow heavily from another in-house effort--the MUMS project.[6] MUMS (for Modular Unified Microprocessor System) is an attempt to define a very general microcomputer system architecture which is capable of supporting a wide variety of processors, system modules (e.g., read/write and read-only memories, direct memory access channels, etc.), and peripheral devices.

Although the current (MUMS III) standard is a more general and powerful design, the MICROBURST bus is based on an earlier version (MUMS II) which is completely adequate for the specific application and has the advantage of using the in-house standard 80 contact circuit cards and edge connectors. Also, two modules constructed for the MUMS II prototype system--a Teletype compatible serial interface card and an interrupt priority card--were immediately available, thereby avoiding unnecessary duplication of effort.

The basic interface function is one of data translation--i.e., generating Burst outputs from binary-coded data and interpreting Burst stream inputs in a computer-compatible format. Also, the ability to perform channel switching operations is an important requirement in Burst systems, as already noted.

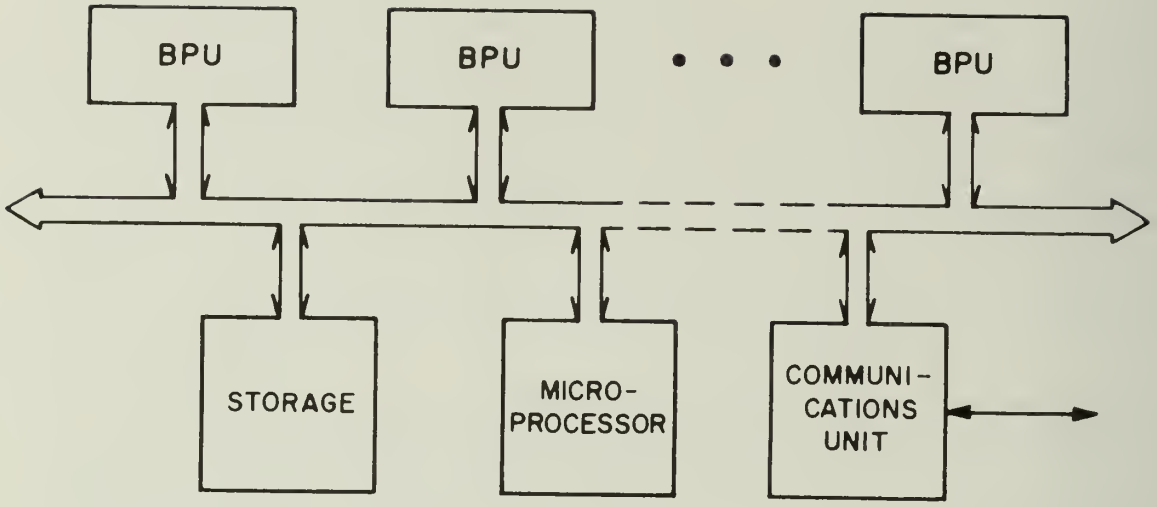


Figure 4a Generalized Model of a Processor-Controlled Burst System

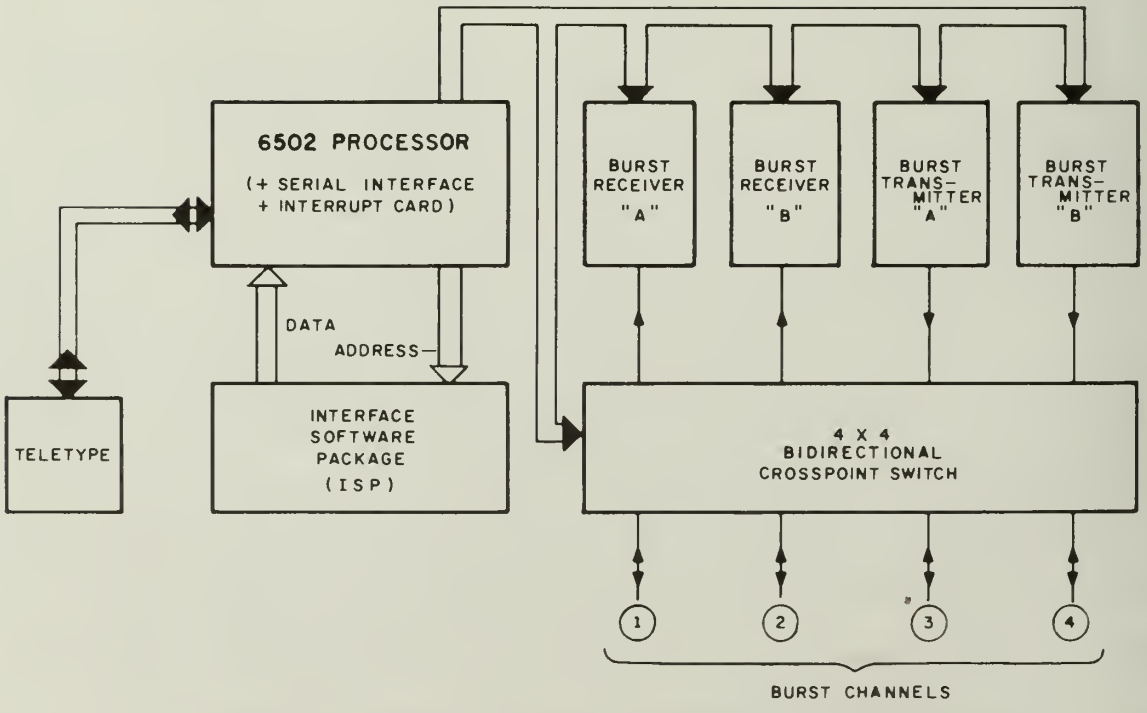


Figure 4b MICROBURST System Block Diagram

Thus, the interface circuits included in the MICROBURST hardware consist of two Burst generators ("transmitters") and two receivers, and a dual 4 x 4 bidirectional crosspoint switch which permits arbitrary interconnections of Burst channels to interface units.

Four Burst channels may thus be interfaced directly with the device: more channels may be supported by extending the crosspoint network. The assignment of channels as inputs and outputs is defined by the state of the switching matrix. A block diagram of the interface hardware is shown in Figure 4 (b).

An ASR Model 33 teletype is used for input and output of commands and data, so front panel controls and indicators are minimized. There are two front-panel switches--one is a latching pushbutton which can be used to switch primary power to external power supplies, and the other is a reset button (momentary contact) which pulls the system reset line low when depressed.

Two indicator arrays on the front panel show the status of the individual interface circuits. Each transmitter and receiver has a LED on the panel which indicates when that particular module is active. Also, a 4 x 4 array of LED's is used to indicate which nodes of the crosspoint array are currently closed.

Six connectors on the front panel provide input and output data to the device. Four 5-pin connectors carry clock and Burst data signals to and from the interface. A single 25-pin socket is wired to accept the Teletype plug. An identical connector is also provided for possible future use as a parallel input/output port.

2.1 Receivers

Since the precision of a Burst-encoded representation increases with the number of frames averaged (up to the limit imposed by the encoding process), it is clear that <u>greater precision requires more time</u> [Eq. 1.2]. It is

therefore undesirable to perform all channel "read" operations to a fixed precision, since this implies an unnecessary reduction in repetition rate when lower precision values will suffice. For this reason the averaging time to be used is a programmed quantity specified as a number of decimal digits (1, 2 or 3) at the beginning of each read operation. This allows a maximum precision of .1 percent, which probably represents the upper limit of practicability in analog-derived Burst systems based on the long averaging times and tight encoder component tolerances necessary.

In order to make the completed interface as useful as possible in future Burst-processing research efforts, it was decided that the receivers should include some capability to "diagnose" a Burst data stream, i.e., to check the stream characteristics during a read operation for illegal or other significant format conditions. Since the receivers function basically as counters, they will provide a correct interpretation of any unary-encoded data, regardless of the specific data format involved.

The notion of the "packed" Burst format has already been introduced (Sec 1.1)--its importance to the hardware simplicity of many processing functions makes it an essential characteristic of the Burst representation.

If we define an unfilled time slot followed by a filled slot as a "SYNC" transition, it is clear that unpacked Burst frames will have two or more SYNC periods in a 10-bit time interval. The occurrence of <u>no</u> SYNC transitions in a frame interval represents a "Loss-of-Sync" (LOS) condition. It should be noted that LOS is not an invalid Burst format--frame values of 0 and 1 (in a 10-slot system) will both result in LOS conditions--in the former case all slots are zeroes and in the latter case all slots are ones.

Although LOS is not in general an error condition, several specialized Burst devices have been proposed which do require that frame synchronization be maintained. In such cases the LOS status signal can be used to detect this undesired condition. To isolate the source of the LOS condition two additional status bits Z (zero frame) and F (full frame) are maintained which flag the appearance of 10 consecutive zero or one data bits, respectively. A fourth status bit, U, flags the occurrence of "unpacked" data occurring in the Burst stream. This condition is sensed by detecting multiple SYNC transitions within a frame interval.

Operation of the receivers is a simple procedure. Address bits A_8-A_{15} are set to the I/O page address, bits A_2-A_7 are set to the appropriate receiver port number, and bits A_0 and A_1 are set to indicate the desired precision of the read operation (see Table 1 for receiver addressing conventions). When a write operation is performed to this address, the counter chains are cleared and the Burst clock and data lines are enabled. Upon completion of the appropriate counting interval (10, 100, or 1000 clock times), further counting is inhibited and an interrupt is requested to the priority card.

Each receiver represents four input ports to the processor for reading data. The address is set up as for the "start" operation just described with bits A_0 and A_1 now specifying the data items to be read.

Each data value is a four-bit quantity with the digits appearing in standard BCD format. The status word is composed of the four flags already described:

Bit	Flag
3	U
2	LOS
1	Z
0	F

A_1	A_0	R/W	
0	0	R	READ STATUS REG
0	1	R	READ L S D
1	0	R	READ M D
1	1	R	READ M S D
0	1	W	INIT READ, PREC=1
1	0	W	INIT READ, PREC=2
1	1	W	INIT READ, PREC=3

Table 1 Receiver Addressing

A_1	A_0	R/W	
0	0	W	LOAD 8 BITS
0	1	W	LOAD 4 BITS
1	X	W	LOAD BASE & START

Table 2 Transmitter Addressing

A_1	A_0	R/W	
0	0	W	LOAD T1 MASK
0	1	W	LOAD T2 MASK
1	0	W	LOAD R1 MASK
1	1	W	LOAD R2 MASK

Table 3 Crosspoint Addressing

These data are typically read and processed as part of the interrupt servicing routine.

2.2 Transmitters

In order to understand the mode of data representation within the Burst transmitters, it is necessary to make an important observation: in any "averaging sequence" (sequence of digits a_i which is used to represent a value x as $x = \frac{1}{n} \sum_{i=o}^{n} a_i$) which minimizes the error in the approximation as n is increased, the digits in the sequence will take on only one of two values--namely the digit values which bracket the value of x. This is a simple and intuitive notion and does not warrant a formal proof here. The idea is simply that no 5's will occur in a sequence which is to represent the value 2.3 with minimum error at each step.

When the fact is realized, then it becomes clear that a minimum-error sequence can be efficiently represented by a bit string (equal in length to the cycle length of the output sequence) and a single digit value. Then the averaging sequence to be produced is defined by the simple rule that a zero value in the bit string specifies that a frame value equal to the stored digit value be output, while a one encountered in the bit string requires that a frame value one greater than the stored value be produced.

A Burst stream is represented in exactly this fashion within the transmitter hardware. Since a "ten-slot" frame is assumed, a four-bit counter is used to hold the "BASE DIGIT." Each bit of a 100-bit recirculating shift register is used to represent a frame value equal to the BASE DIGIT (if the bit is a zero) or a frame value of BASE DIGIT + 1 (if the bit is a one).

The actual unary frames are produced from the frame value counter by straightforward combinational and sequential circuits. As each 10-slot frame is completed, the shift register is circulated one bit position and the frame value counter is adjusted appropriately.

It should be mentioned here that the circuitry used to generate the unary frames is incapable of producing a frame value equal to zero (a full frame is produced in this case). This means that values in the range $0 \leq x < .100$ cannot be represented. However a full decade of values may still be produced ($.100 \leq x < 1.000$), and since the Burst format is inherently a "scaled" representation this was not deemed to be a deficiency.

Thus the use of the transmitter requires only that the 100-bit shift register be loaded with the desired bit pattern and the BASE DIGIT be set. (Transmitter addressing conventions are detailed in Table 2.) Since each shift register bit represents an entire Burst frame, the use of 100 bits in the register implies a Burst output represented to a precision of 10^{-3} which is consistent with the maximum receiver resolution of 3 decimal digits.

Each transmitter has an on-board clock generator which is disabled whenever the shift register is loaded and enabled when a BASE DIGIT is entered.

2.3 Crosspoint Network

The 4 x 4 crosspoint switch has been included to allow efficient switching of data between the four Burst channels supported by the device and the four transmitter/receiver circuits which it incorporates. This array is actually two identically constructed and controlled 4 x 4 networks, with one handling data channels and the other carrying clocking information.

The switches used are CD4066 CMOS analog switches, which are useful because of their low on-resistance, high off-resistance, and bidirectional nature. Four such chips (with four switches per chip) are used in each array. Sixteen flipflops store the control (switches on/off) data for each node, and drive the front-panel switch status lights. Addressing formats for referencing the crosspoint control flipflops are described in Table 3.

Data may be routed through several successive nodes to achieve whatever interconnection pattern is desired. Up to four nodes may be traversed without severe degradation of the signal waveforms, but greater numbers tend to introduce excessive series resistance into the signal path and provide unreliable operation.

2.4 Processor and Support Hardware

The central processor utilized is the MOS Technology Inc. MCS 6502. This is an 8-bit n-channel MOS device which operates off a single supply of +5 volts and has a basic clock rate of 1 MHz. Four particular attributes suggested an advantage for the 6502 over other processors then available. They are, in order of importance:

1) Low Cost--by far the lowest (twenty dollars in unit quantities) of any comparable 8-bit device.
2) Decimal Arithmetic Capability--which simplifies some of the decimal-oriented functions performed by the interface software.
3) Multiple Addressing Modes--this tends to reduce program length for functions such as table-driven routines and string operations.
4) Zero Page Addressing --which leads to a further reduction in program size when all or part of the program workspace can be confined to page zero.

To create a usable computer system from a device of this nature requires that a moderate amount of additional circuitry be added external to the processor chip itself. Naturally, memory must be provided for program and data storage. More chips are required, however, for such purposes as buffering of all address, data, control, and clock lines, and generation of I/O and memory control signals which conform to the protocol defined for the bus structure in use. The addition of more sophisticated capabilities such as hardware interrupt arbiting, direct memory access, single cycle/instruction execution, etc., each requires that external logic be implemented.

Four circuit cards comprise the processing section of the interface. These are a processor card, a memory card, an interrupt priority card, and a serial interface card. The processor and memory cards constitute the actual "computer". Programs are loaded into 2,048-bit ROM's of the 1702 type which are inserted into one of eight sockets on the memory board. Each socket represents a different "page" (256-word segment) of the memory addressing space. The memory address decoding is such that all possible 16-bit addresses map into one of ten physical pages of memory (or the I/O device "page" which is decoded on the processor board.) Of these, eight are ROM's as already mentioned, and two pages are read/write memory used for program workspace. The ROM pages are indicated by a one in address bit 15; the individual ROM is then selected by bits A_8, A_9, and A_{10} (see Table 4). A zero in A_{15} specifies a reference to read/write memory, and A_8 then selects between page "zero" and "one." Address bits A_0-A_7 are used as a common address bus to all memory chips.

A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	
1	X	X	X	X	0	0	0	ROM PAGE F8
1	X	X	X	X	0	0	1	ROM PAGE F9
1	X	X	X	X	0	1	0	ROM PAGE FA
1	X	X	X	X	0	1	1	ROM PAGE FB
1	X	X	X	X	1	0	0	ROM PAGE FC
1	X	X	X	X	1	0	1	ROM PAGE FD
1	X	X	X	X	1	1	0	ROM PAGE FE
1	X	X	X	X	1	1	1	ROM PAGE FF
1	0	1	0	0	0	0	0	I/O PAGE
0	X	X	X	X	X	X	0	RAM PAGE 00
0	X	X	X	X	X	X	1	RAM PAGE 01

Table 4 Memory Space Decoding

A_2	A_1	A_0	R/W	
0	0	1	R	READ UART DATA
0	0	1	W	WRITE UART DATA
1	0	1	R	READ FLAGS

Table 5 Serial Interface Addressing

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
0	0	0	1	0	0	X	X	CROSSPOINT
0	1	0	0	0	0	X	X	TRANS 1
0	1	0	1	0	0	X	X	TRANS 2
0	1	1	0	0	0	X	X	RCVR 1
0	1	1	1	0	0	X	X	RCVR 2
0	0	0	0	0	X	X	X	SERIAL I/O

Table 6 Port Number Assignments

An I/O page is decoded on the processor board and is used to generate signals which control "external" devices. This page is switch-selectable, but should be set to page $A0_{16}$ when used with the initial ISP (Interface Software Package) program.

The interrupt card receives eight "Interrupt Request" lines and generates an interrupt signal and a 3-bit "Priority Acknowledge" word (which can be accessed at location FF_{16} of the I/O page) when any of its inputs are activated. The "Interrupt Request" lines are open-collector and each line may therefore be shared among several interrupt-generating devices, although the interrupt service routine will then be required to identify the interrupt source through polling.

Finally, the serial interface card allows the processor to communicate with serial devices such as Teletypes. The clock rate used is switch-selected at either 110 or 300 baud, and status flags are available which indicate the activity of the UART (Universal Asynchronous Receiver-Transmitter) device. Data and status ports recognized by the serial interface are defined in Table 5.

The overall structure and hardware capabilities of the interface have now been described in sufficient detail to allow one to reprogram the system to suit special needs. The following chapter will briefly describe the circuit details of the current implementation and outline the bus operation so as to allow construction of additional functional units.

3 CIRCUIT DESCRIPTIONS

3.0 Bus Operation

The MUMS II bus, which was modified only slightly for use in the MICROBURST interface, is a collection of signals which permit general communications between a processing unit and memory or peripheral devices.

A diagram of the pinout locations for the various signals is shown in Figure A1.1 of the Appendix. Data is transferred in byte-parallel fashion along eight tristate lines, and sixteen bits of address information are maintained (the processor is the only source of addresses in the MICROBURST system, so these lines are unidirectional). In addition, eight data lines originally defined for possible use with 16 bit processors have been redefined as clock and data lines for the four Burst channels.

Eight lines are used for interrupt requests; the recognized priority level is coded and transmitted to the processor in the three-bit PRIORITY ACKNOWLEDGE field. The I/O address lines (8) merely parallel the low-order address byte and either set of lines may be decoded for address recognition by devices.

Many of the other signals are self-explanatory--two lines for the phase 1 and phase 2 system clocks, a $\overline{\text{RESET}}$ line which when active (low) causes the system to be initialized, a $\overline{\text{INT}}$ line which interrupts the processor when taken low, and nine connections to various power supply voltages and ground.

Also, several of the MUMS II signals are not used in the interface. These include two lines defined for "daisy-chaining" purposes (CHAIN IN, CHAIN OUT) the SYNC line, two direct-memory access signals $\overline{\text{HOLD}}$ and HLDA, the interrupt enable control INTE, and the $\overline{\text{RDY}}$ line.

The remaining signals are those which control the type and timing of bus data transactions. Each bus cycle is identified by four lines as being either a

memory read, memory write, I/O read, or I/O write operation. The corresponding active-low signal names are $\overline{\text{RDM}}$, $\overline{\text{WRM}}$, $\overline{\text{INP}}$, and $\overline{\text{OUT}}$. One of these lines is asserted for each bus cycle and the timing is such that the signal will be activated only after address information is stable on the bus. Thus, address decoding by memory and I/O devices is enabled by these signals.

Data transfers take place during the phase 2 clock cycle; for read operations data must be valid at the falling edge of phase 2, while valid data for processor write operations is indicated by another control signal. This signal, $\overline{\text{WR}}$, is asserted only when the data is valid during write operations, and is used as a strobe to the memory chips and data storage registers in peripheral units.

3.1 Receiver Circuits

Central to the operation of the receivers are two three-digit binary-coded decimal counter chains. One of these chains counts incoming Burst clock pulses to establish the counting interval of 1, 10, or 100 frames, while the other counts Burst data bits to accumulate the stream value. A 7474 dual D-flipflop retains the two low-order address bits present during the last "write" to that receiver, as they specify the precision of the read operation. These bits are used to select the carry output (via a four-input multiplexer) from one of the three digits of the "clock" counter which is used to identify the end of the prescribed counting period. Another 7474 is used to synchronize the starting and stopping of the counters to the Burst clock to insure that no errors will result due to insufficient setup times.

Diagnostic information is obtained by four simple sequential machines each consisting of two flipflops. A 74175 quad D-flipflop is used with combinational circuitry to look for the status conditions described in section 2.1 over each

ten-bit interval. Feedback is used so that any flagged condition is retained until the register is cleared every tenth bit time. Another four-bit register (74194) looks at the frame-by-frame results and retains any flagged conditions until another read operation is initiated.

A four-bit data item from one of the receiver output ports (status and digits 1-3) is selected by two dual 4-to-1 multiplexers and gated onto the data bus when a processor "read" to that receiver is performed.

Port address recognition is accomplished with 74LS136 open-collector exclusive-or gates. This technique is used for address recognition by all devices (except serial I/O) and for detection of I/O references on the processor card.

3.2 Transmitters

The transmitter circuits are built around 100-stage MOS shift registers (Signetics 2510) which store the arrangement of frame values to be output. A 74165 eight-bit shift register is used for parallel-to-serial conversion of data received from the processor, and a four-bit counter (74193) controls entry of either four or eight bits as determined by address bit A_0 (stored in a D-flop).

The BASE DIGIT is stored in a 74191 bidirectional counter, which is initially loaded by the processor. Simple gating forms the proper MODE and CLOCK ENABLE signals from the current and previous (retained in a D-flop) shift register bits to correctly adjust the frame value counter.

A four-bit decoder (74154) is used to generate a ten-bit string from the frame value counter which has the end of the packed bits marked by a low-level output. This string is "scanned" by a sixteen-input multiplexer and decade counter operating from the on-board clock, and converted to packed Burst format

by the D-flop and NOR gates seen on the extreme right of Figure A1.3. Pollover of the decade counter indicates end-of-frame, and is used to clock the shift register and reset the Burst generator.

The on-board clock is composed of cross-coupled monostables to allow independent adjustment of on and off times. This clock should not be operated beyond approximately 10 MHz, as pulse stability becomes poor at this point.

3.3 Crosspoint Switch

The crosspoint switch is composed of eight type CD4066 CMOS analog transmission gates organized as two 4-by-4 arrays (one for Burst data channels, and the other for clock information). The sixteen node control lines for each array are paralleled so that clock and data lines are always switched as pairs. All Burst signals are buffered on the "device" side of the newtork (via 8T97 noninverting drivers) to minimize the effect of switch "on" resistance. Storage for control information is provided by four 4-bit D-registers. Data is loaded from the processor data lines into the register selected by the two low-order address bits. All control bits are set to zero (nodes open) when the system is reset.

The inverted output of each control flipflop is brought to the front panel and used to illuminate the crosspoint status indicators.

3.4 Processor

The MCS6502 processor is used in a fairly minimal configuration since several possible extensions (e.g., non-maskable interrupt, DMA, single-cycle execution) are not used in the interface system. The functions performed by the processor card are address (8T97) and data (8216) buffering, generation of bus control signals, decoding of I/O page references, and interrupt support

functions (generation of INTA and provision of an input port for reading the PACK vector). Also, automatic power-up reset operation is provided by an RC-input Schmitt trigger.

The generation of bus control signals (upper right-hand portion of Figure A1.5) is straightforward--direction of the transfer is defined by the R/W (read/write) signal provided by the processor at the beginning of each cycle, and references to the I/O page are decoded by eight open-collector exclusive-or gates. Proper timing is achieved by the use of a "lock-out" one-shot which inhibits all four control signals until address information is known to be stable (~ 300 ns after the cycle begins).

The write strobe signal ($\overline{WR}$) is generated whenever a write cycle is indicated (R/W = 0), and another lock-out monostable delays assertion of this line until late in phase 2 when data is known to be valid.

A 7430 8-input NAND decodes references to location FF_{16}, and is gated with "I/O Page detected" to generate the interrupt acknowledge (INTA) signal, which also enables the PACK input port.

3.5 Memory

Eight 256-byte "pages" of read-only memory are provided on the memory board, along with two pages of read/write memory and necessary address decoding and data buffering.

Each ROM page consists of a single 1702A UV-eraseable, programmable ROM. The read/write memory is constructed from 2606B 1K chips which are organized as 256 4-bit words--thus two chips are required per page of RAM.

A 74S138 3-to-8 decoder selects the appropriate chip for ROM references, while RAM decoding is performed by the simple gate network seen at the lower left of Figure A1.6.

3.6 Serial Interface

The operation of the serial interface is defined by the MOS LSI "UART" chip (a General Instruments AY-5-1013). This circuit performs all necessary parallel-to-serial and serial-to-parallel conversion, provides proper data framing (start/stop bits, parity, etc.) and has flag bits available which indicate the device status (buffer empty, received data available) and error conditions (buffer overrun, parity error, etc.). Two external clocks are provided (Signetics 556 dual timer) to allow selection of either a 110 or 300 baud data rate. The only other hardware required consists of two output ports (for data and status words), a single data input port, device address decoding, and a small amount of discrete circuitry to interface the UART logic levels to 20 mA current-loop data lines.

3.7 Interrupt Priority

The interrupt card responds to low-level inputs on any of eight interrupt request lines and generates a coded 3-bit word (PACK) which identifies the highest level line activated, as well as issuing an interrupt request to the processor. Eight D-flops latch up any active interrupt request line and feed this forward to a 74148 priority encoder. The "input-active" output of the encoder (EO) is used to set the Interrupt Request flipflop. Upon receipt of an INTA signal from the processor the recognized interrupt latch is cleared and the Request flipflop is reset. Any lower-level interrupts still pending will then be immediately recognized if no higher-level interrupts have since been asserted.

4 SYSTEM SOFTWARE

4.0 General

As with any processor-based system, the MICROBURST interface requires that a suitable operating program be inserted (in the form of programmed read-only memories) into the completed hardware. Of course, the selection of the exact set of functions to be supported by the software depends upon the specific applications for which the device will be used. For this system in particular it is anticipated that the actual system program will be modified and rewritten numerous times to accomodate varying needs. Thus, the initial Interface Software Package (ISP) includes only a basic set of routines intended both to verify hardware operation and to make the system immediately available for general-purpose use. To accomplish this goal, a simple command language incorporating four statement types which allow full user control of the three different interface circuits (transmitters, receivers, and crosspoint network) is interpreted by the ISP. The statements, their defined syntax, and their effects are outlined in the following section.

It must be emphasized here that the current ISP implementation by no means exploits the full capability of the system. Many additional functions can readily be implemented. Some of these possibilities include:

1) Inclusion of software timing facilities to allow channel scanning as well as time-varying Burst outputs.
2) Optional Burst output formats--allowing, say, "vernier-encoder-type" format in addition to the optimally-distributed format currently produced.
3) Routines to perform automatic extraction of stream statistics from AC channels.

4) Self-checking routines to inspect for proper operation of the interface devices (e.g., a complete "health-assessment" program to test all transmitters, receivers, and crosspoint nodes could trivially be included as part of the power-up reset operation).

5) Programs to allow efficient communications between the interface and larger digital processing systems.

The sophistication of the functions performed by the interface is limited only by the desires of the user, and more realistically, by the size of the decoded memory addressing space, which is presently 2,048 8-bit words of read-only memory. It should be noted that the current ISP implementation uses approximately 60 percent of the available memory space.

The software is organized as a number of short routines; an assembly language listing of the complete program is provided in Appendix B. Logically, these routines can be divided into four distinct subsystems based on function. They are 1) syntax analysis and interpretation of commands, 2) BPU driver routines, 3) interrupt servicing (receiver operation), and 4) I/O routines.

Since the current ISP implementation will likely be replaced eventually with a more sophisticated program, the discussion which follows will be more in the vein of a "user's guide" and details of program operation will be ignored.

4.1 ISP Command Language

Four statement types are included which allow full user control of the Burst input, Burst output, and channel switching facilities. Each statement type is indicated by a one-letter "opcode"--allowable opcodes are C, D, R, and W. The first two (Connect and Disconnect) control the operation

of the crosspoint network. Opcode R signifies a Read operation and activates one of the Burst receivers while W denotes a Write command and controls transmitter operation.

4.1.1 Crosspoint Commands

The significance of the C and D commands is straightforward--they are used to either "close" a given node (C) or "open" a node (D) of the crosspoint array. Since the nodes are addressed as pairs X,Y where the columns (X) represent devices (R1, R2, T1, and T2) and the rows (Y) represent data channels (A,B,C, and D), the C and D commands must have both X and Y values present as parameters. Statement format is:

.C X Y

and

.D X Y

where X and Y are selected from the possibilities listed above. (The period preceeding each opcode is a prompting character issued by the ISP and is not a part of the command.) At least one blank must separate the parameter fields and no blanks may occur between the device code (T or R) and the device number (1 or 2). When the input line is terminated by a "RETURN" character the indicated crosspoint operation will be performed if the command syntax was correct, otherwise an error message indicating the first syntax violation encountered will be issued. Another type of error is possible in the case of the C command--this error occurs when an attempt is made to connect two transmitters to the same data channel. In this event the user is asked to resolve the conflict by disconnecting the first transmitter before connecting the second. No check is made to determine if "secondary" transmitter conflicts exist--i.e., transmitters connected to different channels which are in turn connected via intermediate nodes.

4.1.2 Read Command

The R opcode is used to initiate a read cycle by a specified Burst receiver. The format is:

.R X P

where the device (X) is now restricted to either R1 or R2 and the precision of the read is specified in the P field as a number of decimal digits (1,2, or 3). The result of executing an R-command is to clear the counters and enable the clock and data lines in the specified receiver. Burst data must first, of course, be routed to the appropriate receiver via the crosspoint switch.

4.1.3 Write Command

Generation of a Burst output stream having a specified value is accomplished by the W command. The format is:

.W X VAL

where now the device, X, is limited to either of T1 or T2. The VAL field represents the stream value to be generated. Since all Burst data is assumed to represent values normalized to the range $.100 \leq x < 1.000$, the VAL data entered must conform to this range or an error message will result. Correct VAL syntax is:

$$VAL := .n_1 n_2 n_3$$

where n_2 and n_3 are optional and n_1 must not equal zero.

4.2 BPU Driver Routines

The result of the syntax analysis phase of ISP operation is to produce coded values representing the operation type, device and channel involved, etc., which are maintained as global variables used by the driver routines when the command is executed (by typing a "RETURN" character.) There are three different driver routines--one for each type of interface circuit. The cross-

point routine (XPT) performs the necessary manipulations to correctly address specified nodes on the crosspoint switch. An internal map of the crosspoint status is maintained for this purpose, and also for noting the occurrence of "channel conflicts" when connecting transmitters. The receiver driver routine (RCVRSET) is another routine which performs simple address computations to properly address the receiver modules.

The transmitter driver routine (DCOMP) is the most complex of the three and will be briefly described. The primary function of the DCOMP program is to generate the 100-bit sequence which represents the distribution of frame values to be output. It is clear that this amounts to an interpolation of the decimal value n_2n_3 from the values 0 and 100 (we can neglect the most significant digit n_1 since it is effectively handled as an "offset" by the transmitters). The interpolation required is easily implemented in software via the "accumulator rate-multiplication" technique.[7]

Since the interpolation is performed on two-digit decimal quantities the computations are most directly done in the decimal mode. The MCS6502 microprocessor used in the interface can perform decimal arithmetic efficiently on packed two-digit quantities, making it a particularly simple function to implement. After generating and loading the complete 100-bit sequence, the routine loads the base value, n_1, which starts transmitter operation.

4.3 Interrupt Servicing

The receivers signal completion of a read cycle by interrupting the processor. Since an 8-level priority interrupt scheme is used, up to eight interrupt-generating devices can be accommodated without the need for polling during interrupt servicing. The present function of the interrupt routine is simple--identify the interrupting receiver, read value and status data, and format and print the data. Status information is currently printed

as a four-bit binary string although a more sophisticated program could inhibit print unless abnormal conditions were detected. It should be mentioned that reading the PRIORITY ACKNOWLEDGE vector (PACK) is a vital operation during the interrupt sequence (even if only one interrupting device exists within the system) since by this action the INTERRUPT REQUEST flipflop on the interrupt card is reset.

4.4 I/O Routines

Three simple I/O routines are also included in the ISP to perform the elementary functions of reading and writing single characters via the Teletype interface, and outputting strings of characters from various message tables.

5 CONCLUSIONS

The decision to use Burst techniques for a particular application will certainly hinge on the answers to such questions as: is sufficient time available to obtain the desired precision, are the processing functions those for which Burst hardware is simpler than other techniques, and are data channel noise characteristics of a type for which Burst offers an error rate advantage? But it is also true that if interfacing Burst networks to other computing systems proved to be an awkward and expensive proposition, then the attractiveness of this mode of processing would be significantly diminished.

However, the MICROBURST interface shows that this is not the case. A microprocessor-based structure has been used to develop a Burst/weighted-binary interface which is simple and low in cost yet quite versatile.

Although the modular, bus-oriented "functional module" approach has worked well in this system and allows a single structure to be "customized" to a particular application, it is not argued that this is the single "best" architecture. Changing influences such as the increasing performance and decreasing costs of electronic components tend to make any optimal solution a transient one. As fast "bit-sliced" processing elements become more available, for example, it may become desirable to replace much of the special interface logic which now resides in the BPU hardware with a programmed device. In this way some of the flexibility which is necessarily lost as a result of using hardwired logic (fixed frame length, fixed precision limits, etc.) can be recovered.

It is the author's conclusion that the most promising areas for the application of Burst techniques are those where relatively simple arithmetic

operations are to be performed in an on-line fashion on low-frequency, analog-derived data values. Systems of this type are common for providing instrumentation and control of large, slowly-responding devices. In such areas, Burst may be practical as a low-cost, noise-tolerant replacement for analog processing functions. It is felt that the demonstrated feasibility of a general-purpose Burst/binary interface makes such systems a yet more viable alternative.

APPENDIX A:

CIRCUIT DIAGRAMS

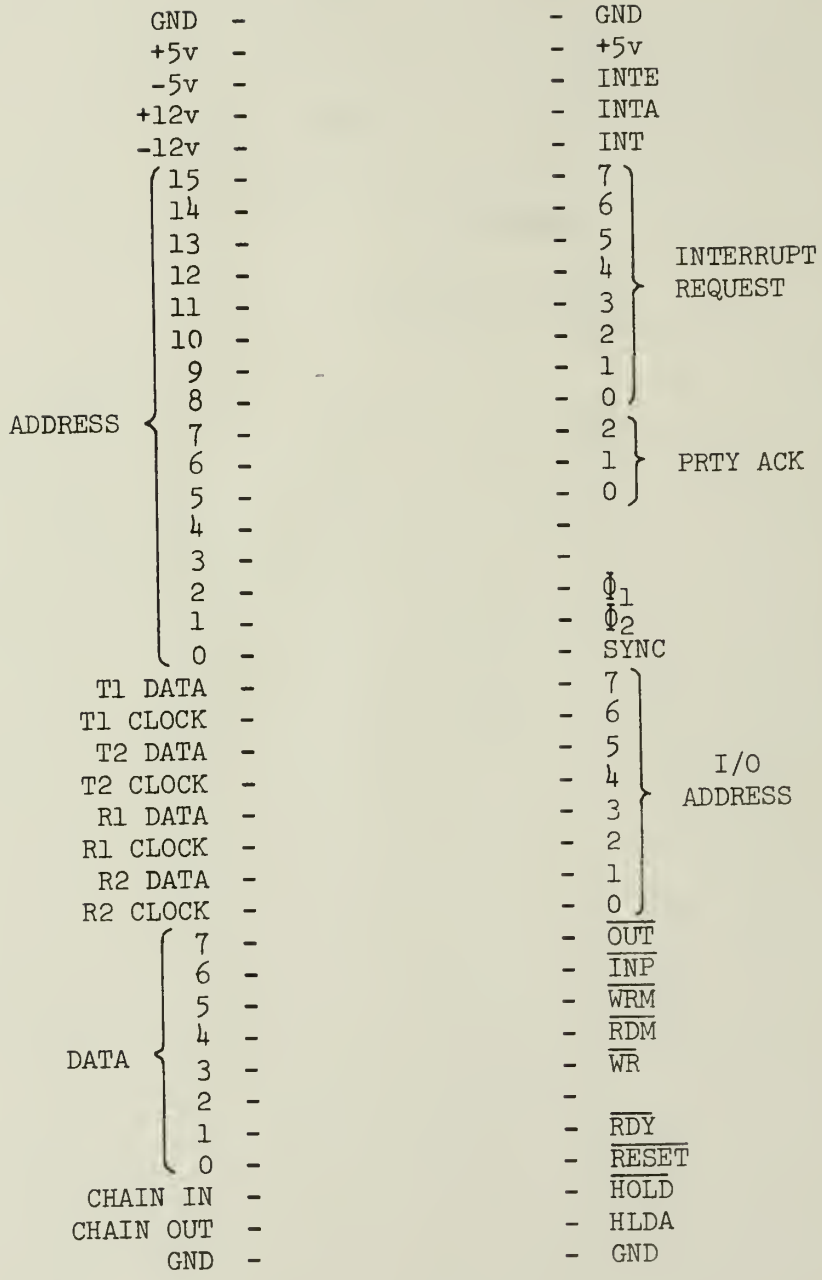


Figure A1.1 Control Bus Pinouts

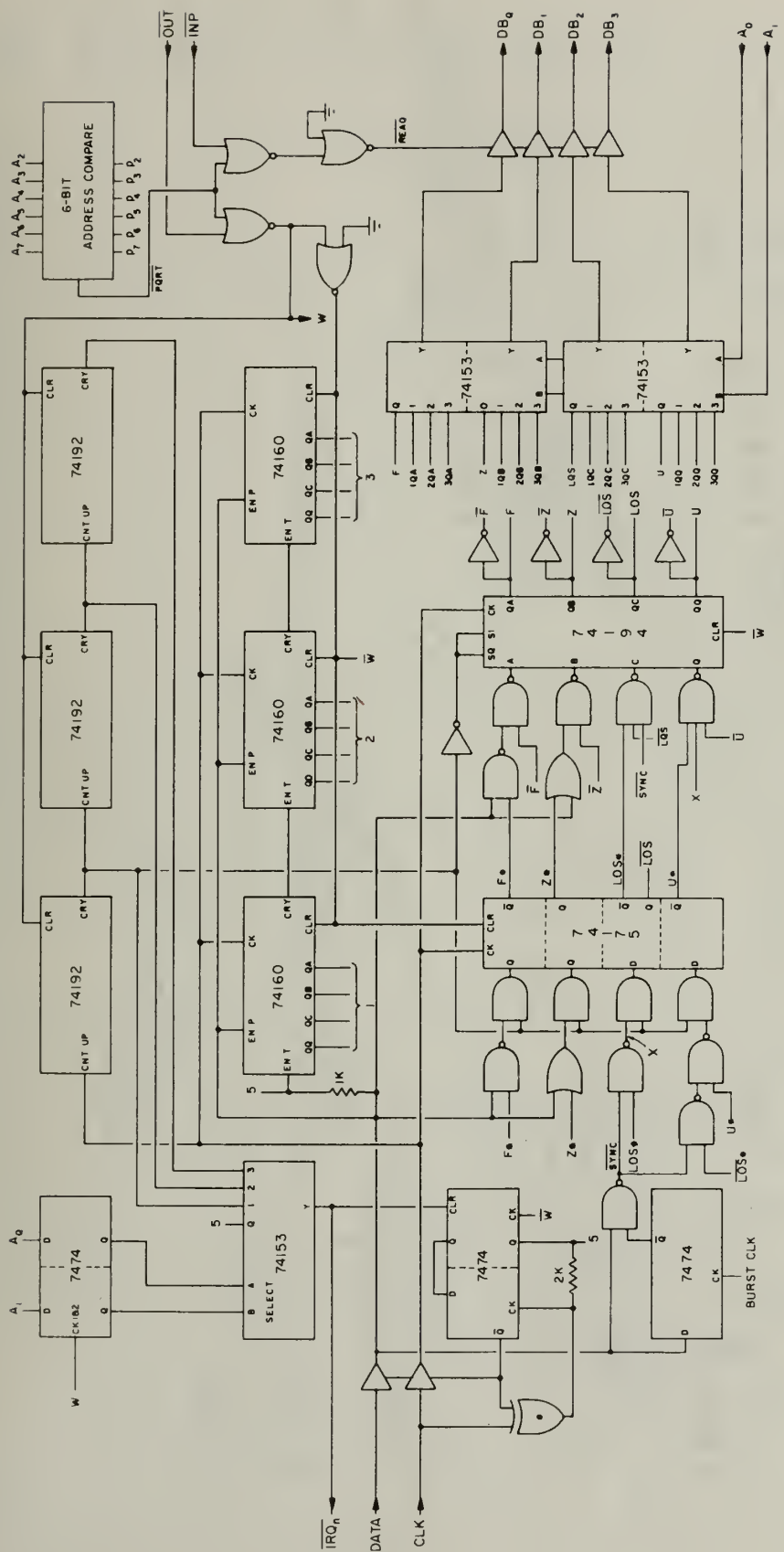


Figure A1.2 Receiver Circuit

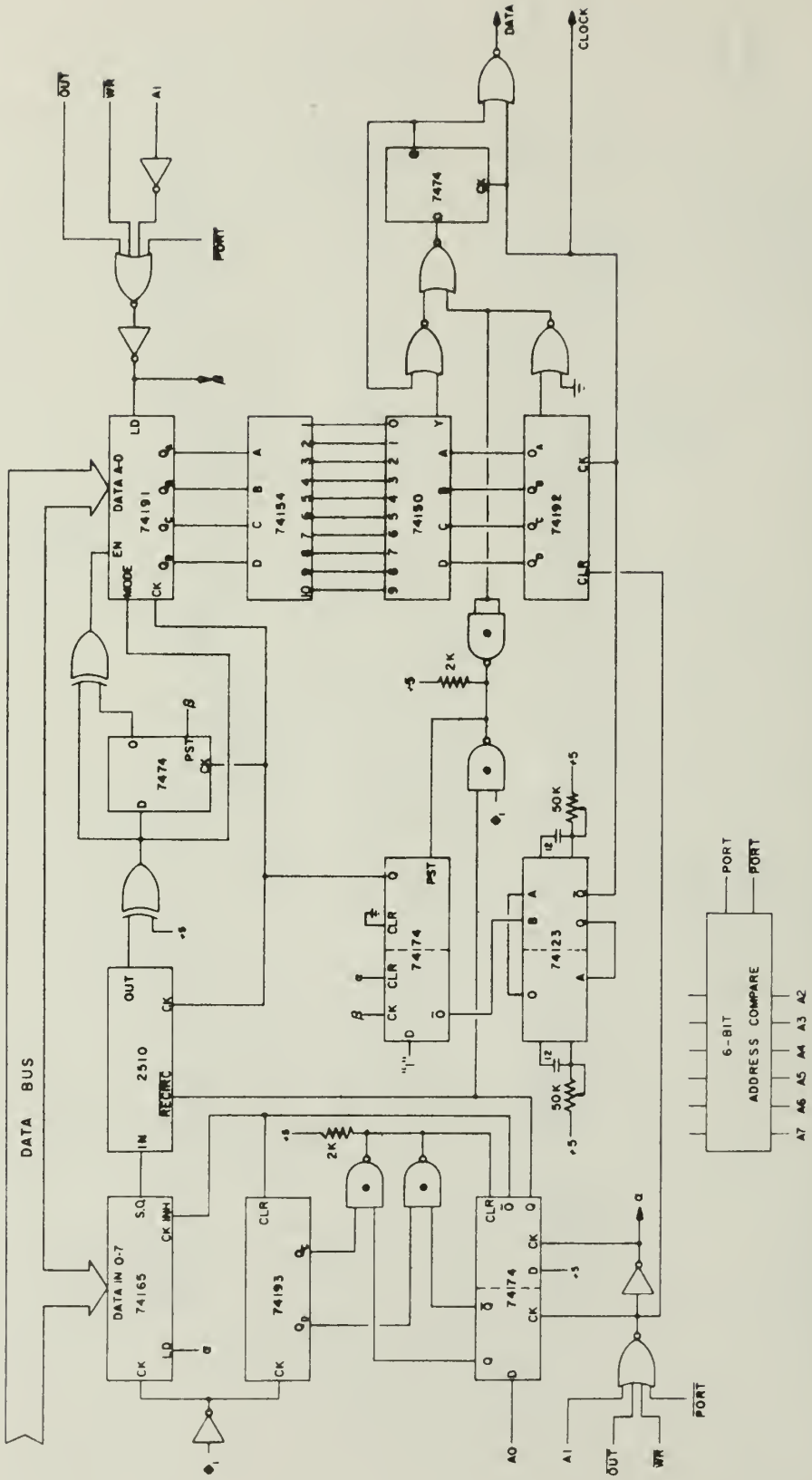


Figure A1.3 Transmitter Circuit

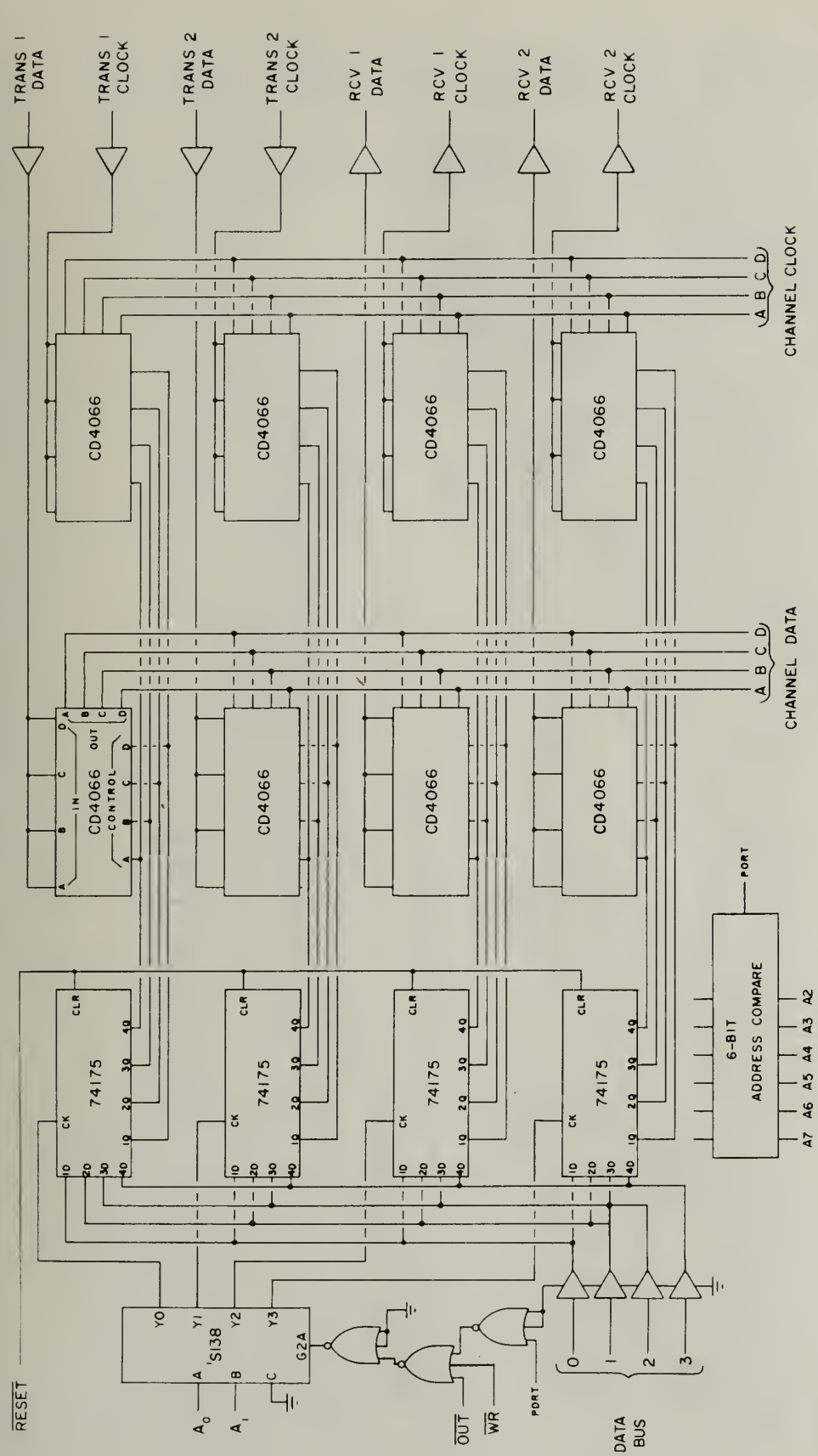


Figure A1.4 Crosspoint Switch Circuit

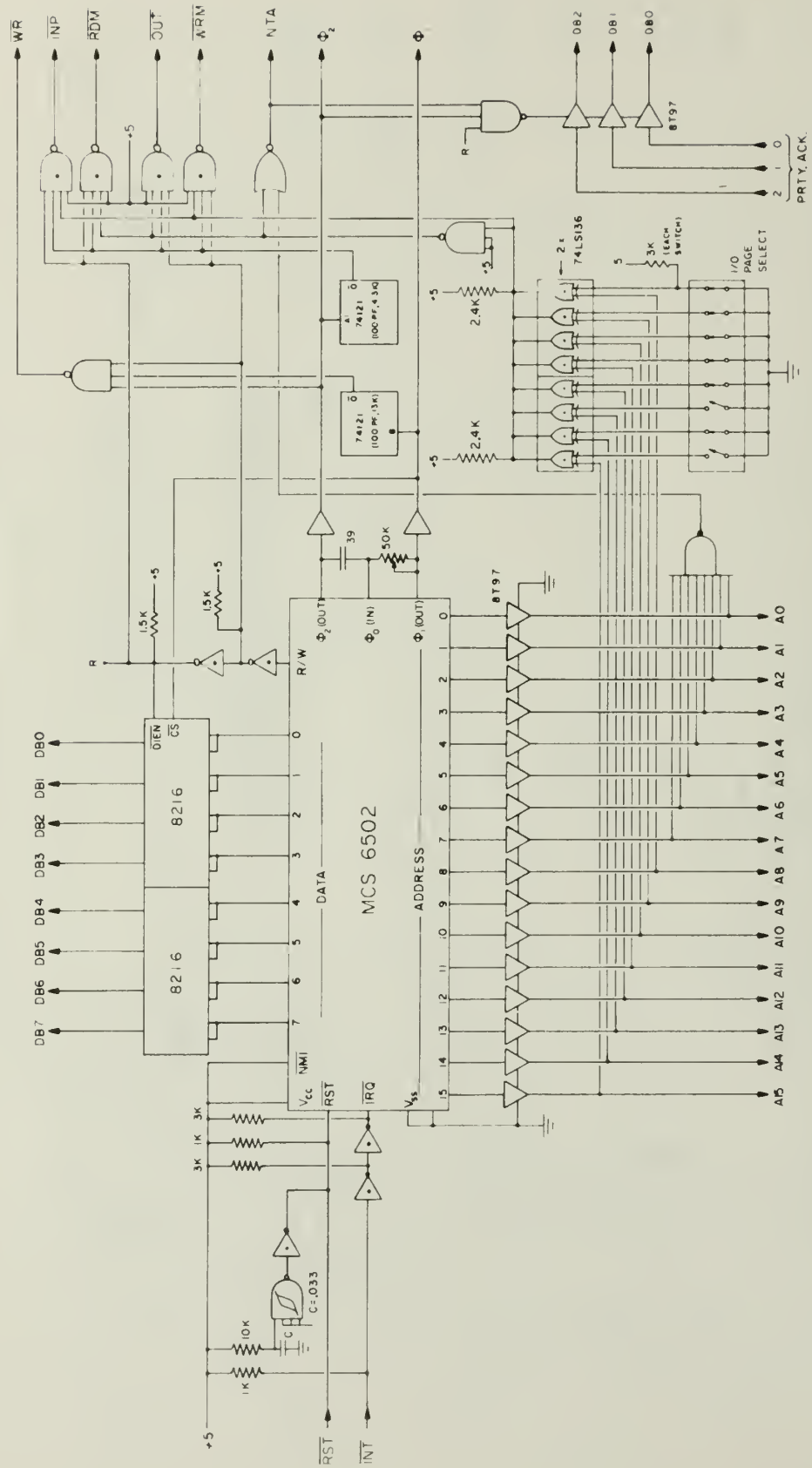


Figure A1.5 Processor Circuit

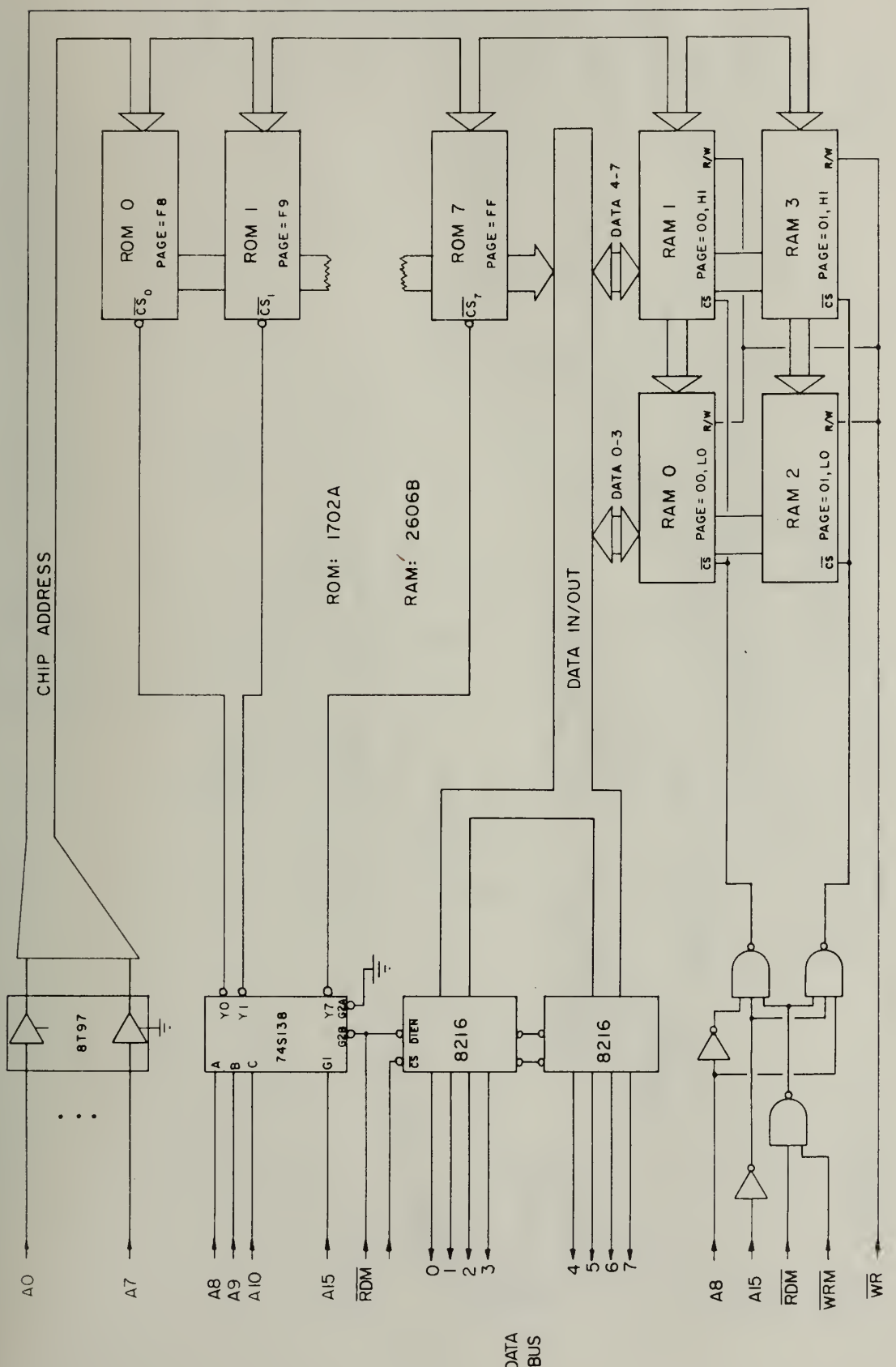


Figure A1.6 Memory Circuit

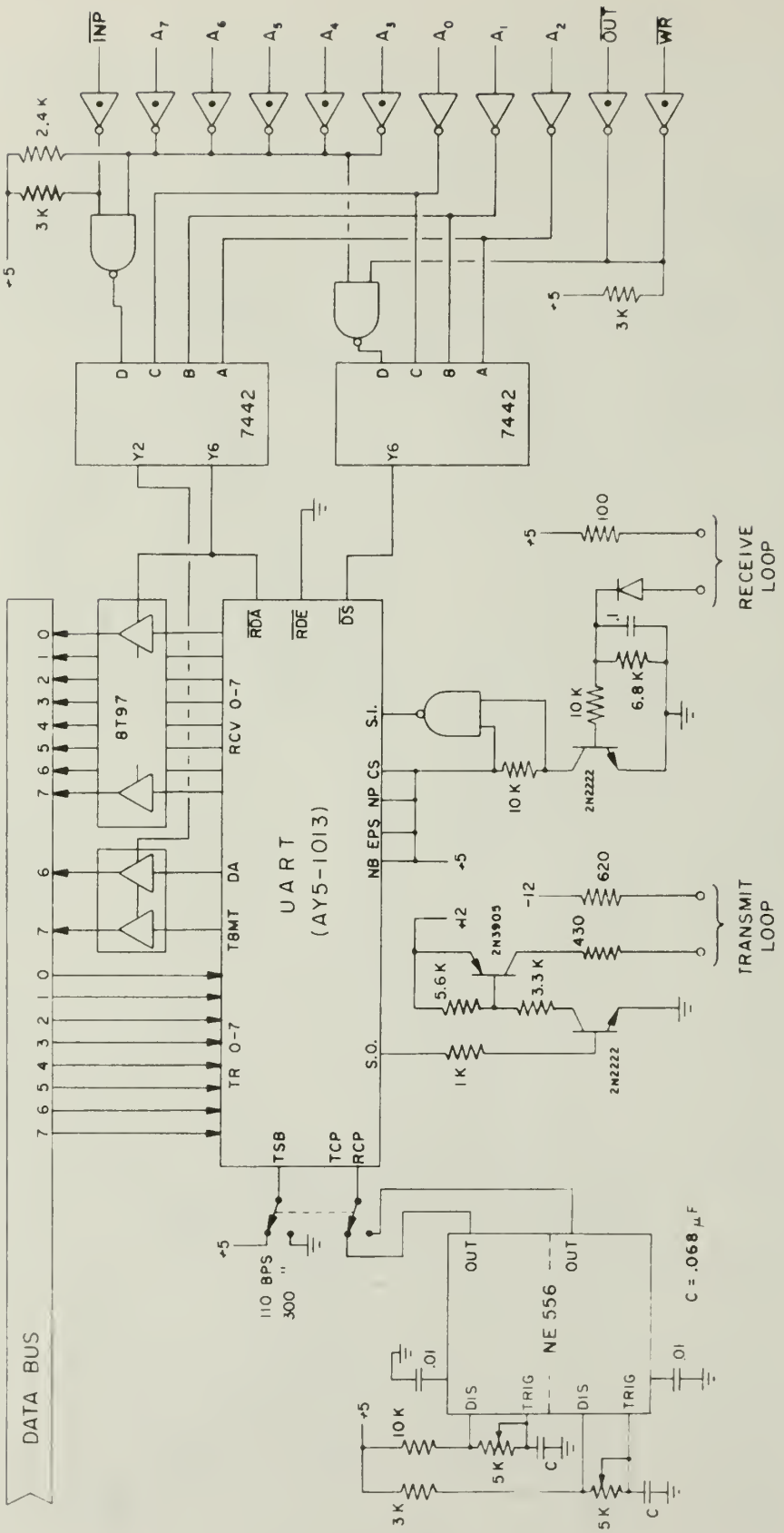


Figure A1.7 Serial Interface Circuit

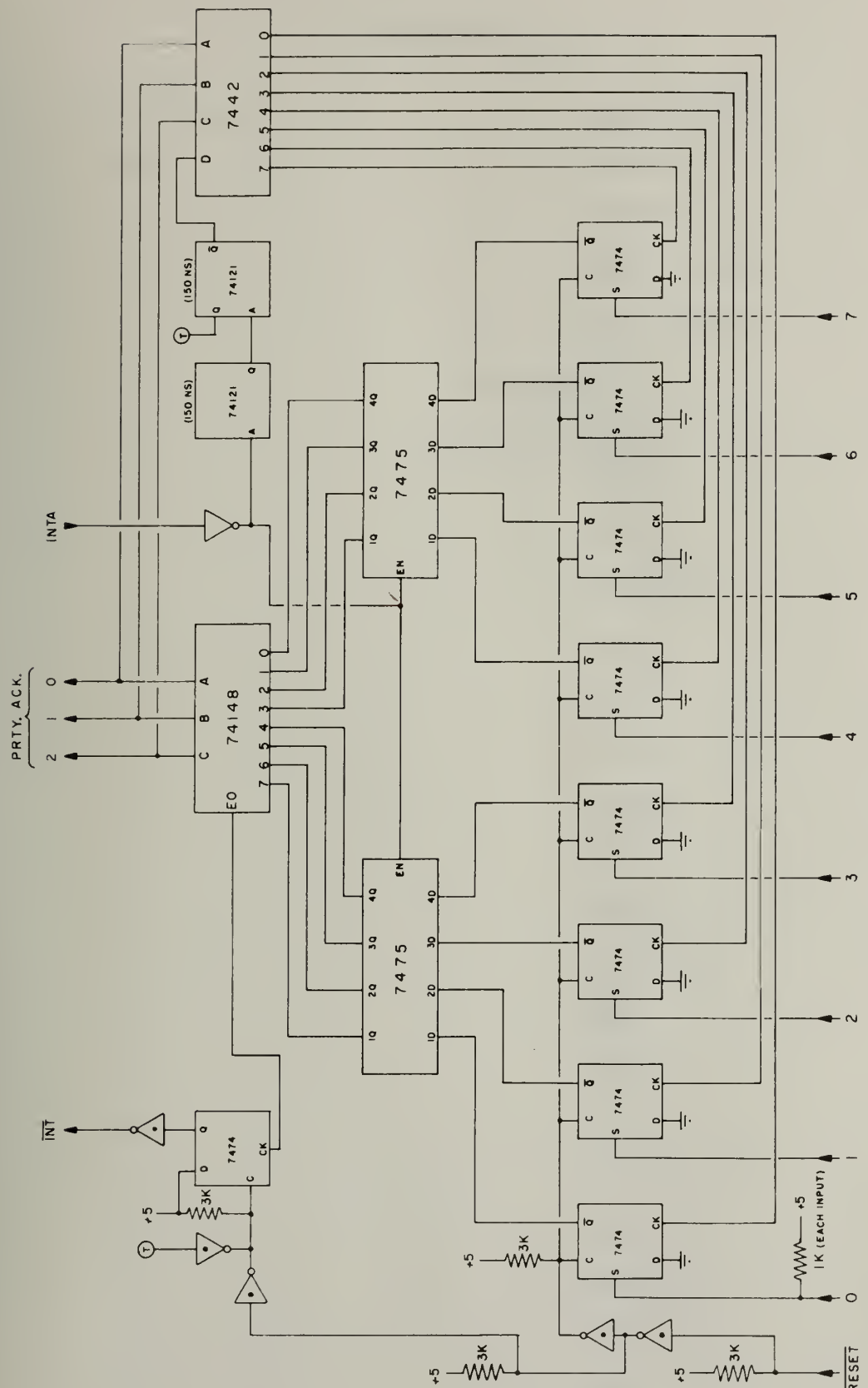


Figure A1.8 Interrupt Priority Circuit

APPENDIX B:

INITIAL ISP LISTING

PAGE ZERO MAP

Hex Address	Variable Name
00:	STATE.LO
01:	STATE.HI
02:	NEXTSTATE.LO
03:	NEXTSTATE.HI
0A:	NUM1
0B:	NUM2
0C:	NUM3
10:	EFLAG
11:	EMSGINDX
12:	PREQZ
15:	OP
16:	DEVICE
17:	CHANNEL
18:	PRECISION
20:	OUTFILE.LO
21:	OUTFILE.HI
30:	MAP
31:	MAP + 1
32:	MAP + 2
33:	MAP + 3
B0:	TEMP
B1:	TEMP + 1
B2:	TEMP + 2
B3:	TEMP + 3

PAGE ZERO MAP (cont'd.)

Hex Address	Variable Name
F0:	PORT
F1:	PVAL
F2:	PVAL*
F3:	SIGNACC
F4:	ACC
F5:	BITS

PAGE = F8

```
      MAIN: LDAI    "."
            JSR     WRITECHAR           ;WRITE PROMPT CHAR
            LDAI    @OPCODE.LO          ;SET STATE = OPCODE
            STAZ    STATE.LO
            LDAI    @OPCODE.HI
            STAZ    STATE.HI
            LDAI    *RESET              ;CLEAR ERROR FLAG
            STAZ    EFLAG
      NEXT: JSR     READCHAR            ;WAIT FOR TTY_ENTRY
            CMPI    "RTN"
            BNE     PO
            JMP     EXEC                ;EXECUTE IF "RETURN"
        PO: BITZ    EFLAG
            BMI     NEXT                ;IGNORE IF ERROR SET
            JMPI    STATE               ;GO TO CHAR ROUTINE

    OPCODE: CMPI    " "
            BNE     P1
            JMP     NEXT                ;SKIP IF BLANK
        P1: CMPI    "C"
            BEQ     FIX
            CMPI    "D"
            BNE     CHECK               ;BRANCA IF NOT XPT OP
       FIX: LDXI    @DEVICE.LO          ;SET NEXTSTATE = DEVICE
            STXZ    NEXTSTATE.LO
            LDXI    @DEVICE.HI
            STXZ    NEXTSTATE.HI
            JMP     OUT
     CHECK: CMPI    "W"                 ;"WRITE" OPCODE?
            BNE     LASTCHANCE
            LDXI    @TRANS.LO           ;SET NEXTSTATE = TRANS
            STXZ    NEXTSTATE.LO
            JMP     OUT
LASTCHANCE: CMPI    "R"                 ;"READ" OPCODE?
            BNE     ERROR1              ;INVALID OPCODE
            LDXI    @RCVR.LO            ;SET NEXTSTATE = RCVR
            STXZ    NEXTSTATE.LO
            LDXI    @RCVR.HI
            STXZ    NEXTSTATE.HI
       OUT: LDXI    @BLANK.LO           ;SET STATE = BLANK
            STXZ    STATE.LO
            LDXI    @BLANK.HI
            STXZ    STATE.HI
            STAZ    OP                  ;STORE OPCODE
            JMP     NEXT                ;NEXT CHARACTER
    ERROR1: LDAI    *SET                ;SET ERROR FLAG
            STAZ    EFLAG
            LDAI    ERR1
            STAZ    EMSGINDX            ;STORE MSG POINTER
            JMP     NEXT                ;NEXT CHARACTER
```

```
BLANK:  CMPI    " "
        BNE     ERROR2              ;ERROR IF NONBLANK
        LDAZ    NEXTSTATE.LO
        STAZ    STATE.LO            ;TRANSFER NEXTSTATE...
        LDAZ    NEXTSTATE.HI        ;...TO STATE
        STAZ    STATE.HI
        JMP     NEXT                ;NEXT CHARACTER
ERROR2: LDAI    *SET
        STAZ    EFLAG
        LDAI    ERR2
        STAZ    EMSGINDX
        JMP     NEXT

DEVICE: CMPI    " "
        BNE     P2
        JMP     NEXT                ;SKIP IF BLANK
   P2:  CMPI    "T"                 ;TRANSMITTER?
        BNE     P3                  ;CHECK IF RCVR
        LDXI    00
        STXZ    DEVICE              ;DEVICE = TRANS
        BPL     P4
   P3:  CMPI    "R"                 ;RECEIVER?
        BNE     ERROR3              ;ERROR IF NOT
        LDXI    02
        STXZ    DEVICE              ;DEVICE = RCVR
   P4:  LDXI    @DEVNO.LO           ;SET STATE = DEVNO
        STXZ    STATE.LO
        LDXI    @DEVNO.HI
        STXZ    STATE.HI
        JMP     NEXT                ;NEXT CHARACTER
ERROR3: LDAI    *SET
        STAZ    EFLAG
        LDAI    ERR3
        STAZ    EMSGINDX
        JMP     NEXT

DEVNO:  LDXI    @BLANK.LO           ;SET STATE = BLANK
        STXZ    STATE.LO
        LDXI    @BLANK.HI
        STXZ    STATE.HI
        CMPI    "2"                 ;DEVICE #2?
        BNE     P5
        INCZ    DEVICE              ;IF SO, ADJUST "DEVICE"
        BPL     P6
   P5:  CMPI    "1"                 ;DEVICE #1?
        BNE     ERROR4              ;ERROR IF NOT
   P6:  LDAI    "C"
        CMPZ    OP                  ;IS OP C OR D?
        BEQ     XPTOP
        LDAI    "D"
        CMPZZ   OP
        BNE     READWRITE           ;BRANCH IF NOT
```

```
    XPTOP:  LDAI    @CHAN.LO            ;SET NEXTSTATE = CHAN
            STAZ    NEXTSTATE.LO
            LDAI    @CHAN.HI
            STAZ    NEXTSTATE.HI
            JMP     NEXT                ;NEXT CHARACTER
READWRITE:  LDAI    "W"
            CMPZ    OP                  ;"WRITE" OP?
            BNE     READ?               ;IF NOT, IT'S A READ
            LDAI    @DECPT.LO           ;SET NEXTSTATE = DECPT
            STAZ    NEXTSTATE.LO
            LDAI    @DECPT.HI
            STAZ    NEXTSTATE.HI
            JMP     NEXT                ;NEXT CHARACTER
    READ?:  LDAI    @PREC.LO            ;SET NEXTSTATE = PREC
            STAZ    NEXTSTATE.LO
            LDAI    @PREC.HI
            STAZ    NEXTSTATE.HI
            JMP     NEXT                ;NEXT CHARACTER
```

PAGE = F9

```
    ERROR4: LDAI    *SET                ;ERROR: INVALD DEVICE NUMBER
            STAZ    EFLAG
            LDAI    ERR4
            STAZ    EMSGINDX
            JMP     NEXT

      PREC: CMPI    " "
            BNE     P7
            JMP     NEXT                ;SKIP IF BLANK
        P7: CMPI    "1"                 ;CHAR < 1?
            BCC     ERROR5              ;ERROR IF SO
            CMPI    "4"                 ;CHAR ≥ 4?
            BCS     ERROR5              ;ERROR IF SO
            ANDI    07
            STAZ    PRECISION           ;STORE PRECISION
            LDAI    @EXEC.LO            ;SET STATE = EXEC
            STAZ    STATE.LO
            LDAI    @EXEC.HI
            STAZ    STATE.HI
            JMP     NEXT                ;NEXT CHARACTER
    ERROR5: LDAI    *SET                ;ERROR: ILLEGAL PRECISION
            STAZ    EFLAG
            LDAI    ERR5
            STAZ    EMSGINDX
            JMP     NEXT
```

```
CHANNEL:  CMPI    " "
          BNE     P8
          JMP     NEXT              ;SKIP IF BLANK
      P8: TAX                       ;FORM CHANNEL MASK
          LDAI    01                ;LOAD MASK BIT
          CPXI    "A"               ;CHAR = A?
          BEQ     MATCH             ;IF SO, EXIT
          ASLA                      ;SHIFT MASK BIT
          CPXI    "B"               ;CHAR = "B"?
          BEQ     MATCH             ;IF SO, EXIT
          ASLA                      ;SHIFT MASK BIT
          CPXI    "C"               ;CHAR = "C"?
          BEQ     MATCH             ;IF SO, EXIT
          ASLA                      ;SHIFT MASK BIT
          CPXI    "D"               ;CHAR = "D"?
          BNE     ERROR6            ;IF NOT, ERROR
  MATCH:  STAZ    CHANNEL           ;STORE MASK
          LDAI    @EXEC.LO          ;SET STATE = EXEC
          STAZ    STATE.LO
          LDAI    @EXEC.HI
          STAZ    STATE.HI
          JMP     NEXT              ;NEXT CHARACTER
 ERROR6:  LDAI    *SET              ;ERROR: NO SUCH CHANNEL
          STAZ    EFLAG
          LDAI    ERR6
          STAZ    EMSGINDX
          JMP     NEXT

  TRANS:  CMPI    " "
          BNE     P9
          JMP     NEXT              ;SKIP IF BLANK
     P9:  CMPI    "T"               ;DEVICE A TRANS?
          BEQ     p10               ;IF NOT, ERROR
          JMP     ERROR3
    P10:  LDAI    @DEVNO.LO         ;SET STATE = DEVNO
          STAZ    STATE.LO
          LDAI    @DEVNO.HI
          STAZ    STATE.HI
          LDAI    00                ;DEVICE = TRANS
          STAZ    DEVICE
          JMP     NEXT              ;NEXT CHARACTER

   RCVR:  CMPI    " "
          BNE     P11
          JMP     NEXT              ;SKIP IF BLANK
    P11:  CMPI    "R"               ;DEVICE A RCVR?
          BEQ     P12               ;IF NOT, ERROR
          JMP     ERROR3
    P12:  LDAI    @DEVNO.LO         ;SET STATE = DEVNO
          STAZ    STATE.LO
          LDAI    @DEVNO.HI
          STAZ    STATE.HI
          LDAI    02                ;DEVICE = RCVR
          STAZ    DEVICE
          JMP     NEXT              ;NEXT CHARACTER
```

```
DECPT:  CMPI    " "
        BNE     P13
        JMP     NEXT            ;SKIP IF BLANK
  P13:  CMPI    "."             ;CHAR A PERIOD?
        BNE     ERROR7          ;IF NOT, ERROR
        LDAI    00              ;INTIALIZE NUM1-3
        STAZ    NUM1
        STAZ    NUM2
        STAZ    NUM3
        LDAI    @NUM1.LO        ;SET STATE = NUM1
        STAZ    STATE.LO
        LDAI    @NUM1.HI
        STAZ    STATE.HI
        JMP     NEXT            ;NEXT CHARACTER
ERROR7: LDAI    *SET            ;ERROR: INVALID VAL SYNTAX
        STAZ    EFLAG
        LDAI    ERR7
        STAZ    EMSGINDX
        JMP     NEXT

 NUM1:  CMPI    "0"             ;NUM1 = 0?
        BEQ     ERROR8          ;IF SO, UNNORMALIZED
        CMPI    "1"             ;NUM1 < 1?
        BCC     ERROR7          ;IF SO, ERROR
        CMPI    "9"+1           ;NUM1 > 9?
        BCS     ERROR7          ;IF SO, ERROR
        ANDI    07
        STAZ    NUM1            ;STORE IT
        LDAI    @NUM2.LO        ;SET STATE = NUM2
        STAZ    STATE.LO
        LDAI    @NUM2.HI
        STAZ    STATE.HI
        JMP     NEXT            ;NEXT CHARACTER
ERROR8: LDAI    *RESET          ;ERROR: INVALID VAL SYNTAX
        STAZ    EFLAG
        LDAI    ERR8
        STAZ    EMSGINDX
        JMP     NEXT

 NUM2:  CMPI    " "             ;CHAR A BLANK?
        BNE     P14
        LDAI    @EXEC.LO        ;IF SO, READY TO GO
        STAZ    STATE.LO        :SET STATE = EXEC
        LDAI    @EXEC.HI
        STAZ    STATE.HI
        JMP     NEXT


PAGE = FA

   P14: CMPI    "0"             ;CHECK FOR VALID NO.
        BCC     BADNUM
        CMPI    "9"+1
        BCS     BADNUM
        ANDI    0F
```

```
            STAZ    NUM2            ;STORE IT
            LDAI    @NUM3.LO        ;SET STATE = NUM3
            STAZ    STATE.LO
            LDAI    @NUM3.HI
            STAZ    STATE.HI
            JMP     NEXT
BADNUM:     JMP     ERROR7

  NUM3:     CMPI    " "
            BEQ     TERM            ;DONE IF BLANK
            CMPI    "0"             ;IN RANGE?
            BCC     NOPE            ;IF NOT, ERROR.
            CMPI    "9"+1
            BCS     NOPE
            ANDI    0F
            STAZ    NUM3            ;STORE IT
  TERM:     LDAI    @EXEC.LO        ;READY TO GO
            STAZ    STATE.LO        ;SET STATE = EXEC
            LDAI    @EXEC.HI
            STAZ    STATE.HI
            JMP     NEXT            ;NEXT CHARACTER
  NOPE:     JMP     ERROR7
```

PAGE = FB

```
   XPT:     LDXZ    DEVICE          ;GET DEVICE CODE
            LDAI    "C"
            CMPZ    OP              ;OP A CONNECT?
            BEQ     CONN
  DCON:     LDAZ    CHANNEL         ;IF NOT GET MASK
            EORI    FF              ;INVERT IT
            ANDZ,X  MAP             ;"AND" WITH CURRENT MASK
            STAZ,X  MAP             ;STORE NEW MASK
            STA,X   XPTSWITCH       ;WRITE TO XPT
            RTS
  CONN:     LDAZ    CHANNEL         ;OP IS CONNECT
            CPXI    02              ;TRANS CONNECT?
            BCC     CHEK            ;IF SO, BRANCH
            ORAZ,X  MAP             ;"OR" IN NEW MASK
            STAZ,X  MAP             ;STORE NEW MASK
            STA,X   XPTSWITCH       ;WRITE TO XPT
            RTS
  CHEK:     TXA
            EORI    01              ;REFERENCE OTHER TRANS
            TAY
            LDAZ    CHANNEL         ;GET NEW MASK
            AND,Y   MAP             ;"AND" WITH OTHER T-MASK
            BNE     CONFLICT        ;NO CONFLICT IF ZERO
            LDAZ    CHANNEL
            ORAZ,X  MAP             ;"OR" IN NEW MASK
            STAZ,X  MAP             ;STORE NEW MASK
            STA,X   XPTSWITCH       ;WRITE TO XPT
            RTS

                                    ;
```

```
CONFLICT:  LDAI     MSG
           JSR      OUTPUT          ;WRITE CONFLICT MSG
           LDAI     "RTN"
           JSR      WRITECHAR
           LDAI     "LF"
           JSR      WRITECHAR       ;RESET TTY TO LHS
           RTS

RCVRSET:   LDXZ     DEVICE          ;GET DEVICE CODE
           LDAI     60              ;LOAD R1 PORT NO.
           CPXI     03              ;IS IT R1?
           BCC      RCVR1
           LDAI     70              ;IF NOT, LOAD R2 PORT NO.
RCVR1:     ORAZ     PRECISION       ;FORM PROPER ADDRESS...
           TAX                      ;...FROM PORT NO. AND PREC
           STA,X    I/O             ;WRITE TO RCVR
  INF:     SEC                      ;WAIT FOR INTERRUPT
           BCS      INF
DCOMP:     LDAZ     DEVICE          ;GET DEVICE NO.
           BEQ      TR1             ;BRANCH IF T1
           LDAI     50              ;T2 PORT NO.
           STAZ     PORT
           BPL      INIT
  TR1:     LDAI     40              ;T1 PORT NO.
           STAZ     PORT
 INIT:     LDAZ     NUM2            ;GET MIDDLE DIGIT
           ASLA
           ASLA
           ASLA
           ASLA                     ;MOVE TO H.O. FOUR
           ORAZ     NUM3            ;FORM PACKED BCD
           STAZ     PVAL            ;KEEP FOR INTERP.
           LDAI     99
           SEC
           SED                      ;MODE=DECIMAL
           SBCZ     PVAL
           CLC
           ADCI     01              ;100 - PVAL
           STAZ     PVAL*           ;KEEP FOR INTERP.
           LDAI     00
           STAZ     ACC
           STAZ     SIGNACC         ;INIT FOR INTERP.
           LDXI     0D
           LDYZ     PORT
BLOOP:     DEX
           BEQ      EXIT
           JSR      INTRPSTP        ;CALL INTERP ROUTINE
           LDAZ     BITS            ;GET RESULTS
           STA,Y    I/O             ;WRITE TO TRANS
           JMP      BLOOP
```

```
      EXIT:  JSR      INTRPSTP          ;LAST TIME
             INY                        ;PORT=LOAD 4
             LDAZ     BITS              ;GET RESULTS
             STA,Y    I/O               ;WRITE 4 TO TRANS
             LDAZ     NUM1              ;GET BASE DIGIT
             INY                        ;PORT=BASE DIGIT
             STA,Y    I/O               ;LOAD BASE DIGIT
             CLD                        ;MODE=BINARY
             RTS

  INTRPSTP:  TXA
             PHA                        ;SAVE INDEX REG.
             LDXI     09
        NEXT: DEX
             BEQ      DONE
             BITZ     SIGNACC           ;CHECK SIGN
             BMI      NEG
         POS: CLC
             ROLZ     BITS              ;STRINGBIT=0
             LDAZ     ACC
             SEC
             SBCZ     PVAL              ;FORM NEW ACC
             STAZ     ACC               ;SAVE IT
             BCS      NEXT
             LDAI     FF
             STAZ     SIGNACC           ;CHANGE SIGN ACC
             BMI      NEXT              ;UNCONDITIONAL
         NEG: SEC
             ROLZ     BITS              ;STRINGBIT=1
             LDAZ     ACC
             CLC
             ADCZ     PVAL*             ;FORM NEW ACC
             STAZ     ACC               ;SAVE IT
             BCC      NEXT
             LDAI     00
             STAZ     SIGNACC           ;CHANGE SIGN ACC
             BPL      NEXT              ;UNCONDITIONAL
        DONE: PLA
             TAX      ;RESTORE          INDEX REG.
             RTS
```

PAGE = FC

```
         INT: PLA
             PLA
             PLA                        ;DESTROY RTN ADDRESS
             LDA      PREQ              ;READ PRTY. ACK. VECTOR
             ANDI     01
             STAZ     PREQZ             ;STORE IT
             BEQ      R1                ;BRANCH IF LEVEL Ø
             LDYI     73                ;SET PORT = RCVRZ
             BPL      READ
```

```
      R1:  LDYI     63                ;SET PORT = RCVR1
    READ:  LDXI     04                ;LOOP INDEX
  INLOOP:  DEX
           BMI      EXIT
           LDA,Y    I/O               ;READ RCVR DATA
           ANDI     0F
           ORAI     30                ;CONVERT TO ASC11
           STAZ,X   TEMP              ;STORE IT
           DEY
           BPL      INLOOP
    EXIT:  LDAI     MSG2              ;PRINT HEADER
           JSR      OUTPUT
           LDXZ     PREQZ             ;GET PREQ VECTOR
           INX
           TXA
           ORAI     30                ;MAKE IT ASC11
           JSR      WRITECHAR         ;WRITE IT
           LDAI     MSG3
           JSR      OUTPUT            ;PRINT ":b:"
           LDXI     04                ;LOOP INDEX
 OUTLOOP:  DEX
           BEQ      THRU
           LDAZ,X   TEMP              ;FETCH RCVR DATA
           JSR      WRITECHAR         ;PRINT IT
           JMP      OUTLOOP
    THRU:  LDAI     MSG4              ;PRINT STATHEADER
           JSR      OUTPUT
           LDXI     05                ;LOOP INDEX
   SLOOP:  DEX
           BEQ      DUN
           LSRZ     TEMP              ;INSPECT STATUS BIT 0
           BCS      ONEOUT            ;BRANCH IF A ONE
           LDAI     30                ;LOAD "0"
           JSR      WRITECHAR         ;PRINT IT
           BCC      SLOOP
 ONEOUT:   LDAI     31                ;LOAD "1"
           JSR      WRITECHAR         ;PRINT IT
           BCS      SLOOP
     DUN:  LDAI     "RTN"             ;RESET TTY
           JSR      WRITECHAR
           LDAI     "LF"
           JSR      WRITECHAR
           LDA      PREQ
           NOP
           CLI                        ;CLEAR INTERRUPT FLAG
           JMP      MAIN              ;NEXT COMMAND


PAGE = FF

WRITECHAR: BIT      TTY_STATUS        ;TTY BUSY?
           BPL      WRITECHAR         ;IF SO, LOOP
           STA      TTY               ;WRITE ACC.
           RTS
```

```
READCHAR:  BIT      TTY_STATUS          ;TTY DATA AVAILABLE?
           BVC      READCHAR            ;IF NOT, LOOP
           LDA      TTY                 ;READ DATA
           ANDI     7F
           JSR      WRITECHAR           ;ECHO IT
           CMPI     "RTN"               ;IF RTN, WRITE LF TOO
           BNE      FINIS
           LDAI     "LF"
           JSR      WRITECHAR
           LDAI     "RTN"
  FINIS:   RTS

 OUTPUT:   ASLA                         ;MULT INDEX BY Z
           TAX
           LDA,X    MSGTBL              ;GET LO STRING ADDRESS
           STAZ     OUTFILE.LO
           INX
           LDA,X    MSGTBL              ;GET HI STRING ADDRESS
           STAZ     OUTFILE.HI
           LDYI     00                  ;IN IT STRING INDEX
  PLOOP:   LDAZ,Y   OUTFILE.LO          ;GET STRING CHAR
           BMI      OUTT                ;CHAR = TERM?
           JSR      WRITECHAR           ;IF NOT, WRITE IT
           INY                          ;INCREMENT STRING INDEX
           BPL      PLOOP
   OUTT:   RTS

   EXEC:   CMPI     "RTN"               ;SKIP IF NOT RETURN
           BEQ      GO
           JMP      NEXT
     GO:   BITZ     EFLAG
           BPL      DOIT                ;EXECUTE IF NO ERROR
           LDAZ     EMSGINDX            ;PRINT ERROR MESSAGE
           JSR      WRITECHAR
           LDAI     "LF"
           JSR      WRITECHAR
           JSR      WRITECHAR
           JMP      MAIN                ;NEXT COMMAND
   DOIT:   LDAZ     OP
           CMPI     "K"                 ;SEE IF XPT OP
           BCS      RD/WR               ;BRANCH IF NOT
  XPTOP:   JSR      XPT                 ;DO XPT FUNCTION
           LDAI     "LF"                ;RESET TTY
           JSR      WRITECHAR
           JMP      MAIN
  RD/WR:   CMPI     "W"                 ;SEE IF "WRITE" OP
           BEQ      WR                  ;BRANCH IF SO
           JMP      RCVRSET             ;RCVR ROUTINE
     WR:   JSR      DCOMP               ;CALL DCOMP
           LDAI     "LF"
           JSR      WRITECHAR           ;RESET TTY
           JMP      MAIN                ;NEXT COMMAND
```

```
SYSINIT:  LDXI     FF              ;INIT STACK POINTER
          TXS
          CLC
          CLD                      ;MODE = BINARY
          LDAI     00
          STA      T1,BITS         ;TURN OFF T1 & T2
          STA      T2,BITS
          STA      XPT             ;CLEAR XPT SWITCH
          STA      XPT + 1
          STA      XPT + 2
          STA      XPT + 3
          STAZ     MAP             ;CLEAR XPT MAP
          STAZ     MAP + 1
          STAZ     MAP + 2
          STAZ     MAP + 3
          LDAI     "CR"
          JSR      WRITECHAR       ;RESET TTY
          LDAI     "LF"
          JSR      WRITECHAR
          CLI
          JMP      MAIN            ;WAIT FOR COMMAND

VECTORS:  RES.LO   B0              ;RESET AND INTERRUPT
          RES.HI   FF              ;VECTORING
          INT.LO   00
          INT.HI   FC
```

LIST OF REFERENCES

1. Poppelbaum, W. J., Appendix I to "A Practicability Program in Stochastic Processing," Department of Computer Science, University of Illinois, March, 1974.

2. Bracha, E., "BURSTCALC (A BURST CALCulator)," Report UIUCDCS-R-75-769, Department of Computer Science, University of Illinois, October, 1975.

3. Mohan, P. L., et al., "Performance Evaluation of the Digital AM Receiver," Report UIUCDCS-R-75-757, Department of Computer Science, University of Illinois, April, 1975.

4. Mohan, P. L., "The Application of Burst Processing to Digital FM Receivers," Report UIUCDCS-R-76-780, Department of Computer Science, University of Illinois, January, 1976.

5. Poppelbaum, W. J., "Application of Stochastic and Burst Processing to Communication and Computing Systems," Proposal, Department of Computer Science, University of Illinois, July, 1975.

6. Catlin, R. W., "MUMS: A Modular Unified Microprocessor System," Report UIUCDCS-R-76-809, Department of Computer Science, University of Illinois, July, 1976.

7. Peatman, J. B., The Design of Digital Systems, McGraw-Hill Co., New York, 1972.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: UIUCDCS-R-76-812	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): A MICROPROCESSOR-CONTROLLED INTERFACE FOR BURST PROCESSING	5. TYPE OF REPORT & PERIOD COVERED: Master's Thesis
	6. PERFORMING ORG. REPORT NUMBER: UIUCDCS-R-76-812
7. AUTHOR(s): Pleva, Robert M.	8. CONTRACT OR GRANT NUMBER(s): N000 14-75-C-0982
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Office of Naval Research, 219 South Dearborn Street, Chicago, Illinois 60604	12. REPORT DATE: July 1976
	13. NUMBER OF PAGES: 70
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report): Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Burst processing
Unary encodings
Microprocessor-based systems
Serial data transmission

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

As part of the research effort investigating the properties and applications of Burst processing, the question of compatibility with the more usual weighted-binary data representations has been considered. The MICROBURST system is a microprocessor-controlled, general-purpose interface system capable of operating in a multichannel Burst environment. System structure, programming details, and support software are discussed.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601 |

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-R-76-812	2.	3. Recipient's Accession No.

4. Title and Subtitle

A MICROPROCESSOR-CONTROLLED INTERFACE FOR BURST PROCESSING

5. Report Date: July, 1976

6.

7. Author(s): Robert M. Pleva

8. Performing Organization Rept. No. UIUCDCS-R-76-812

9. Performing Organization Name and Address

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

10. Project/Task/Work Unit No.

11. Contract/Grant No. N000 14-75-C-0982

12. Sponsoring Organization Name and Address

Office of Naval Research
219 South Dearborn Street
Chicago, Illinois 60604

13. Type of Report & Period Covered: Master's Thesis

14.

15. Supplementary Notes

16. Abstracts

As part of the research effort investigating the properties and applications of Burst processing, the question of compatibility with the more usual weighted-binary data representations has been considered. The MICROBURST system is a microprocessor-controlled, general-purpose interface system capable of operating in a multichannel Burst environment. System structure, programming details, and support software are discussed.

17. Key Words and Document Analysis. 17a. Descriptors

Burst processing
Unary encodings
Microprocessor-based systems
Serial data transmission

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages 70
	20. Security Class (This Page UNCLASSIFIED	22. Price

FORM NTIS-35 (10-70)

USCOMM-DC 40329-P71